KPMG

Rapport

Bistand til datateknisk analyse ifm. rettstvist

13.12.21

Sammendrag

KPMG har bistått Advokatfirmaet Simonsen Vogt Wiik (SVW) med datatekniske analyser i forbindelse med en rettslig tvist mellom SVWs klient Magnus Granath og Craig Stephen Wright, representert ved Wikborg Rein Advokatfirma (WR). KPMG har vurdert filene i datagrunnlaget ut fra metadata, innhold og formattering, samt vurdert de av filene som karakteriserer som kildekode opp mot den offentlig tilgjengelige Bitcoin-kildekoden.

For filene som ikke kategoriserer som kildekode, observerer KPMG blant annet inkonsistens i metadata for en rekke av filene, tilsynelatende manglende metadata, bruk av fonter med copyright fra en senere dato enn dateringen av dokumentene, og bruk av programvare som først ble tilgjengelig flere år etter filens angivelige opprettelsestidspunkt og siste endringsdato.

For de filene som er overlevert fra Wikborg Reins prosesskriv som er kategorisert som kildekode, kan ikke KPMG verifisere at disse filene er av en opprinnelsesdato før offentlig tilgjengelig kildekode. KPMG observerer inkonsistens i innholdet i enkelte av kildekode-filene som er vanskelig å forklare på annen måte enn at de har blitt opprettet eller modifisert i tiden etter publisering av offentlig tilgjengelig kildekode.

KPMG vurderer det som sannsynlig at flere av filene i datamaterialet er endret slik at de fremstår å være opprettet tidligere enn de faktisk er.

1155

Innholdsfortegnelse

1155

1 Innledning

1.1 Bakgrunn

KPMG mottok 6. september 2021 en forespørsel fra Advokatfirmaet Simonsen Vogt Wiig (SVW) om å bistå med datatekniske analyser for å vurdere en rekke filer i forbindelse med en rettslig tvist mellom SVWs klient Magnus Granath og Craig Stephen Wright, representert ved Wikborg Rein Advokatfirma. KPMG signerte engasjementsavtale med SVW den 09.10.2021 for bistand til datateknisk vurdering.

1.2 Mandat

Følgende mandat ble gitt KPMG for utførelse av oppdraget ved kontraktsinngåelse:

(a) *Mottak og analyse av elektronisk lagret informasjon for det formål å vurdere ektheten av datafilene, særlig da dateringen av dem og om de er originaler eller manipulasjoner av andre tilgjengelige filer.*
(b) *Utarbeide en sluttrapport som beskriver analyse og konklusjon, og som kan benyttes i den videre rettstvisten.*
(c) *Evt. møte som partsvitne/sakkyndig i retten*

Det ble i et statusmøte den 12.11.2021 spesifisert at SVW ønsker bistand til analyse og vurdering av 73 spesifikke bilag fra Wikborg Reins prosesskriv. Disse bilagene og alle deres vedlegg er listet opp med en markering «X» i kolonnen «Til analyse» i *Vedlegg 1: Filoversikt mottatte filer*.

1.3 Forbehold og avgrensninger

KPMGs oppdrag har vært avgrenset til å gjøre datatekniske vurderinger av datagrunnlaget utlevert av oppdragsgiver i den tilstand de er mottatt av KPMG. Det har ikke vært en del av KPMGs mandat å ta stilling til, eller vurdere innholdet i eller bakgrunnen for søksmålet mellom partene. KPMG tar forbehold om at ny informasjon som ikke er kjent for oss kan påvirke våre vurderinger.

Rapporten er utarbeidet for oppdragsgiver i henhold til inngått engasjementsavtale. Enhver distribusjon eller bruk av rapporten utover dette må avklares med KPMG i forkant.

1.4 Gjennomføring

Etter sikring og mottak av data har KPMG analysert filene i sin datalab på KPMG-Huset i Oslo. Oppdraget er utført av et tverrfaglig team med nødvendig kompetanse og ekspertise for å kunne svare ut de ulike tekniske aspektene knyttet til datagrunnlaget. For en kort CV på alle involverte ressurser, se *Vedlegg 30: Kort CV*.

2 Metode

2.1 Sikring av data

Datagrunnlaget for KPMGs vurderinger ble mottatt gjennom SVWs dataportal «Simonsen Vogt Wiig SVW Connect» den 14. oktober 2021. Datagrunnlaget er opplyst å være hentet fra Aktørportalen før det deretter er prosessert inn på SVWs egen dataportal. Filene ble lastet ned i ZIP-format og deretter prosessert i KPMGs analyseverktøy Relativity og FTK Forensic.

KPMG har i tillegg til dette også sikret dataene direkte på en ekstern disk fra Aktørportalen gjennom SVWs innlogging den 18.november 2021. Dette ble gjort for å kontrollere og evt. verifisere om det hadde skjedd endringer i filene i forbindelse med SVWs opprinnelige nedlasting fra Aktørportalen og opplasting til egen portal. Basert på hash-verdier for de enkelte dokumentene ble datamaterialet fra Aktørportalen verifisert å være identisk med de dokumentene som ble tilgjengeliggjort via "Simonsen Vogt Wiig SVW Connect".

Oversikt over datamaterialet som er tilgjengeliggjort for KPMG og som danner grunnlaget for KPMGs analysearbeid fremkommer av *Vedlegg 1: Filoversikt mottatte filer*. Noen av filene inneholder ytterligere filer som vedlegg.

I tillegg har KPMG lastet ned eller produsert 60 filer som referansefiler. En fullstendig oversikt over disse finnes i *Vedlegg 2: Filoversikt referansefiler.*

2.2 Analyse

Alle datafiler er prosessert i KPMGs eDiscovery verktøy Relativity i Forensic Toolkit (FTK) samt EnCase Forensic. Relativity er en dataplattform som prosesserer og indekserer alle datafiler slik at de gjøres søkbare og kan analyseres basert på innhold og metadata. FTK og EnCase er forensic-verktøy som også kan benyttes for å analysere innhold og metadata. FTK og EnCase gir i tillegg mulighet for mer inngående dybdeanalyser av datamaterialet.

Dokumentene er i tillegg analysert i applikasjonen som korresponderer med filformatet, for eksempel er doc- og docx-filer analysert i Microsoft Word, og PDF-dokumenter er analysert med Adobe Acrobat og Adobe Reader, osv.

2.2.1 Om metadata

Metadata er «data om data» og kan si noe om livsløpet til en datafil eller dokument. Eksempel på metadata er tidsstempel (dato, klokke) for opprettelse eller siste endring, brukernavn på forfatter eller hvilket program som har generert filen. Metadata er lagret i det enkelte filformatet og kan også vises i den enkelte applikasjonen (for eksempel Word eller Acrobat). Metadata vil normalt sett ikke være en del av filens innhold som vises ved en utskrift av filen.

Metadata genereres fra ulike kilder, blant annet vil det filsystemet som filen er lagret på, ha metadata som viser blant annet tidsstempler for opprettelse og siste endring. I tillegg kan de ulike applikasjonene lagre sine metadata i selve filen. Eksempler på dette er

Microsoft Word eller Acrobat som for eksempel kan lagre metadata som viser navn på forfatter, tidspunkt for opprettelse, endring og utskrift.

Endring av innholdet i et dokument/datafil kan endre metadata. Siden metadata beskriver noe av datafilens livsløp, kan denne informasjonen si noe om tidspunkt for når filen er behandlet, hvem som har behandlet den og hvordan den er behandlet. Metadata vil normalt ikke vise hva som eventuelt er endret av innhold i filen.

Flytting eller kopiering av datafiler fra en lagringsenhet til en annen, for eksempel via en minnepinne eller som et vedlegg til epost, kan endre metadata knyttet til filsystemets tidsstempler. Blant annet vil en datafil som kopieres fra en lagringsenhet til en annen få oppdatert tidsstempelet for opprettelse og vil være lik tidspunktet for når kopieringen ble utført. Tidsstempel for siste lagring vil imidlertid ikke endres og i dette tilfellet vil man oppleve at tidsstempel for opprettelse er nyere enn tidspunkt for siste lagring.

Filtypens metadata vil imidlertid ikke endres ved flytting eller kopiering. Filtypens metadata vil for eksempel være tidspunkt for opprettelse og endringer som er registrert av den aktuelle applikasjonen, for eksempel Microsoft Word, Open Office eller Adobe Acrobat.

Metadata kan relativt enkelt slettes eller endres med forsett av en bruker, og det finnes mange tilgjengelige verktøy for dette. For eksempel finnes det verktøy som sletter alle metadata i et Word-dokument før dette sendes som vedlegg til epost. Dette kan for eksempel gjøres for å unngå at metadata viser endringer som er gjort i teksten i de tilfellene man har valgt funksjonen for sporing av endringer. Slike endringer vil være lagret som en del av filens metadata. Andre verktøy lar brukeren endre metadata slik at filen kan fremstå med et konstruert livsløp.

Tidsstempler settes ut fra datamaskinens klokke. For at tidsstemplene skal være korrekte betinger det at enheten som benyttes til opprettelse eller endring av dokumenter/filer er konfigurert med korrekt dato, klokke og tidssone. Normalt vil en PC være koblet opp mot Internett og med automatisk synkronisering av dato og klokke. Det er allikevel enkelt å endre tidsinnstillinger på en PC gjennom å sette dato og klokke til en selvvalgt verdi, samt å slå av funksjon for automatisk synkronisering av kalender og klokke. Hvis en PC benytter en feilinnstilt klokke eller kalender, vil den feilaktige tidsangivelsen bli reflektert i metadata.

Inkonsistens i metadata kan tyde på at filen er manipulert for at den skal fremstå med at annet livsløp enn det faktiske.

For de dokumentene som KPMG har blitt bedt om å vurdere, har KPMG analysert dokumentenes metadata. I hovedsak er det analysert metadata knyttet til de ulike filformatene (f.eks. MS Word, Open Office og Adobe PDF), ikke filsystemets metadata da filene antas ikke å være sikret fra opprinnelig lagringssted, men flyttet mellom ulike lagringssystemer.

2.2.2 Om innholds- og formateringsanalyse

KPMG har i tillegg til analyse av filenes metadata også analysert filene i datagrunnlaget fra et innholds- og formateringsperspektiv. Analyser som faller inn under denne kategorien kan være analyser knyttet til en fils bruk av fonter, papirstørrelse og marger, samt tekst- og formatteringsforskjeller mellom bilagene og eventuelle referansefiler.

Analyser av fonter er særlig relevant for PDF-filer da dette er filer som kan inneholde innebyggede fonter med metadata om fontene brukt i filen. Dette er informasjon som vil være innebygget i filen. Fonters metadata inneholder gjerne en datert copyright, som

derfor kan brukes til å si noe om filens alder. En analyse av bruk av fonter i andre dokumenttyper fra Open Office og Microsoft Word kan også gjøres for å vurdere når innhold kan ha blitt produsert basert på benyttet font eller fontfamilies versjon og utgivelsesdato.

2.2.3 Om kildekode

Kildekode er en samlebetegnelse på ulike syntakser brukt i ulike programmeringsspråk, og lagres nesten i alle tilfeller som «rene» tekstfiler, dvs. tekstfiler uten noen form for formattering. Filtypen vil ofte variere fra programmeringsspråk til programmeringsspråk. I programmeringsspråket C++ benyttes ofte filtypene .cpp og .h.

I programmeringsspråk som C++ vil kildekoden gå gjennom en prosess som kalles *kompilering* før en datamaskin er i stand til å kjøre programmet. I denne prosessen oversettes kildekoden (som er lett lesbar og redigerbar for mennesker) til kjørbar kode som er formatet datamaskiner bruker for å kjøre programmer.

Det er veldig lite metadata som følger med rene tekstfiler som utgjør kildekoden, og det benyttes ofte i dag *versjonskontrollsystemer* for å spore endringer på tvers av en kodebase for enklere å kunne håndtere endringer og fikse eventuelle feil. Kildekoden som KPMG har mottatt som vedlegg til Wikborg Reins prosesskriv har ikke vært lagt til i et slikt versjonskontrollsystem.

KPMG har gjort observasjoner av forskjellene mellom kildekoden fra prosesskrivet til WR og den første offisielle kodebasen for Bitcoin som er tilgjengeliggjort for nedlasting her: https://web.archive.org/web/20130524071850/http://www.zorinaq.com/pub/bitcoin-0.1.0.rar. KPMG har fått bekreftet fra SVW at dette er å anse som et korrekt sammenligningsgrunnlag, og ønsker også å påpeke at denne koden er en senere utgave av 0.1.0 (se detaljert beskrivelse i kapittel 3.3.2), noe som har stor betydning for analysen.

Det er viktig å understreke at KPMG *ikke* har gjort en vurdering om kildekoden som er oversendt gjennom WR sitt prosesskriv er fullstendig i form av at den lar seg kompilere til et funksjonelt program uten andre filer. Våre undersøkelser har vært begrenset til å observere forskjellene i innholdet i oversendt kildekode (linje for linje), og om den inneholder informasjon som kan datere oversendt kildekode før den første offisielt tilgjengelige kildekoden for Bitcoin.

2.3 Verktøy

KPMG har benyttet følgende hovedverktøy i vår analyse:

- ✓ AccessData Forensic Toolkit (FTK versjon 7.1.0.290)
- ✓ Relativity 11
- ✓ EnCase Forensic v. 8.11

I tillegg har KPMG brukt følgende programvarer for å analysere metadata og innhold:

- ✓ Adobe Acrobat Pro 2017, inkludert Adobe Preflight
- ✓ Open Office 2.4, inkludert «Writer»
- ✓ Didier Stevens oledump.py [1]
- ✓ QPDF 10.4.0 [2]

[1] Hentet fra https://blog.didierstevens.com/programs/oledump-py/

[2] Hentet fra https://qpdf.sourceforge.io/, 10.12.21.

- ✓ MuPDF [3]
- ✓ FontForge – 20201107 [4]

Kildekoden har blitt analysert med kodesammenligningsverktøyet Beyond Compare 4.

Analyse av kompilert programkode (.exe) har vært gjort med følgende verktøy:

- ✓ Hex Editor Neo 6.54
- ✓ PE Checksum
- ✓ Detect It Easy (DiE) v3.03

[3] Hentet fra https://mupdf.com/index.html, 10.12.21.

[4] Hentet fra https://fontforge.org/en-US/downloads/ og https://www.linuxfromscratch.org/blfs/view/svn/xsoft/fontforge.html, 10.12.21.

3 Observasjoner

I denne seksjonen vil vi redegjøre for alle observasjoner av de 73 bilagene vi har blitt bedt om å analysere. Observasjonene vil presenteres nedenfor gruppert etter kategoriene *Overordnede observasjoner, Filer* og *Kildekode*.

For en fullstendig oversikt over alle filer som er analysert, sekvensielt etter Wikborg Reins bilagsnummerering, refererer vi til alle rader med en markering «X» i kolonnen «Til analyse» i *Vedlegg 1: Filoversikt mottatte filer*.

3.1 Overordnede observasjoner

3.1.1 Vurdering av filenes opphav

Alle dokumentene KPMG har mottatt fra partene i saken er hentet fra Aktørportalen. Ved å sammenlikne filenes MD5-hash fra filene vi mottok fra SVWs datarom og senere direkte fra Aktørportalen, ser vi at dette er de samme filene. KPMG anser derfor datagrunnlaget å være de filene/dokumentene som Wikborg Rein har lastet opp i Aktørportalen med sitt prosesskriv og alle dets bilag med vedlegg.

3.1.2 Filnavn

På et generelt grunnlag observerer KPMG at alle Wikborg Reins bilag er levert som vedlegg til prosesskrivet. Alle bilagene er levert med filnavn som refererer til filens innhold og datering, som vi antar ikke er filenes originale filnavn. I tillegg har filene mottatt et digitalt stempel med bilagsnummer.

Bilagene har igjen et vedlegg som skal representere beviset i sin originale form. Dette er gjort gjennomgående for alle bilag, men med to unntak (Bilag 82 og Bilag 87). I likhet med bilagene antar vi at også vedleggene har mottatt nye filnavn, som vil tilsi at filene ikke er i deres originalform. KPMG anser på grunn av dette datagrunnlaget som forurenset da vi må anta at endringer (i filnavn) har skjedd men at vi ikke kan stadfeste om vedleggene er endret utover dette.

KPMG har gjennomført ulike tekniske analyser for å vurdere metadata og innhold for de dokumenter og filer som er fremlagt i Wikborg Reins prosesskriv av 27.august 2021.

3.2 Filer

3.2.1 Metadata fra referansefiler

For å vurdere metadata til bilagene har KPMG først analysert metadata til tre referansefiler.

Tabell med metadata fra FTK for referansefilene:

Filnavn	MD5-hash	Date created	Date modified	Creator	Producer
bitcoin.pdf	d56d71ecadf2137be09d8b1d35c6c042	24.03.2009 18:33:15 (2009-03-24 17:33:15 UTC)		Writer	OpenOffice.org 2.4
SSRN-id3440802.pdf	5e210ce3003ffaed204b7b2076c2fc92	24.03.2009 18:33:15 (2009-03-24 17:33:15 UTC)	21.05.2008 20:43:08 (2008-05-21 18:43:08 UTC)	Writer	OpenOffice.org 2.4
bitcoin-draft.pdf	b7026c5b02de23871fc1d80a49e087b	03.10.2008 21:49:58 (2008-10-03 20:49:58 UTC)		Writer	OpenOffice.org 2.4

3.2.1.1 Bitcoin.pdf

Beskrivelse: Dette er et offisielt whitepaper tilgjengelig fra bitcoin.org [5].

Filtype: Denne filen er et PDF-dokument.

Opprettet: Fra filens metadata (Adobe XMP metadata) fremgår det at filen ble opprettet 24.03.2009 kl. 11:33:15 lokal tid, i tidssone UTC – 06:00.

Sist endret: Filen innehar ingen dato for sist endret.

Producer: Av metadataene fremgår det at filen ble laget med OpenOffice versjon 2.4.

Creator: Filen er generert fra «Writer», med en PDF-versjon 1.4, fra Acrobat 5-serien. Denne Acrobat serien ble tilgjengelig fra November 2001 [6].

Andre metadata: Utover dette inneholder filen svært lite metadata.

Se *Vedlegg 3: Metadata «bitcoin.pdf»* for detaljer.

[5] Lastet ned fra https://bitcoin.org/en/bitcoin-paper, 11.11.2021

[6] Hentet fra seksjonen *Notes fra* https://www.loc.gov/preservation/digital/formats/fdd/fdd000122.shtml#:~:text=PDF%201.4%20was%20published%20in,A%20families%20of%20ISO%20standards 02.12.2021.

3.2.1.2 SSRN-id3440802.pdf

Beskrivelse:	Dette er et dokument som er publisert på SSRN.com med Craig Wrights navn den 22.august 2019 [7].
Filtype:	Denne filen er et PDF-dokument.
Opprettet:	Fra SSRN.com står det at filen ble skrevet 21. august 2008. Fra filens metadata ser vi fra FTK at filen ble opprettet 24.03.2009 kl. 11:33:15 lokal tid, i tidssone UTC – 06:00. Dette samsvarer med opprettelsesdato for det originale bitcoin-whitepaperet, offentlig tilgjengelig for nedlastning fra bitcoin.org. Fra XMP-metadataene knyttet til filen, fremgår det imidlertid at filen ble opprettet 24.01.2008 kl. 11:33:15 lokal tid, i tidssone UTC - 06:00. Det er derfor ikke samsvar mellom filens metadata som fremkommer i FTK og XMP-metadataene. Dette kan indikere at XMP-metadataene er endret og tilbakedatert. Vi observerer samtidig at selv om opprettelsesdatoen i XMP-metadataene er over ett år tidligere enn filens opprettelsesdato i FTK, er tidsstemplingen nøyaktig den samme kl. 11:33:15, i samme tidssone UTC- 06:00.
Sist endret:	Videre ser vi at filens metadata for sist endring er 21.05.2008 kl. 18:43:08 lokal tid, i tidssone +01:00. Dette samsvarer med filens XMP-metadata. Begge disse tidsstemplene er i tidssone +01:00, som er en annen tidssone enn filens opprettelsesdato.
Producer:	PDF-filen ble laget via OpenOffice versjon 2.4. Ser vi på filens XMP-metadata for opprettelse, som viser den 24.01.2008, så er dette før Open Office versjon 2.4 ble lansert den 27.03.2008 [8]. Filens XMP-endringsdato er imidlertid etter denne datoen.
Creator:	Filen er generert fra «Writer», med en PDF-versjon 1.6, fra Acrobat 7-serien. Til sammenlikning ble «bitcoin.pdf» generert som en PDF-versjon 1.4, fra Acrobat 5-serien.
Andre metadata:	PDF-filen inneholder mer metadata enn «bitcoin.pdf», som eksempelvis keywords, author, title med mer. Se *Vedlegg 4: Metadata «SSRN-id3440802.pdf»* for detaljer. Dette er metadata som man kan legge inn direkte ved hjelp av en programvare som eksempelvis Adobe Acrobat Pro, eller med Open Office Writer Vi observerer også at filen har XMP-metadata attributter fra Photoshop, til forskjell fra «bitcoin.pdf».

[7] Lastet ned fra https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3440802, 11.11.2021.

[8] Hentet fra https://wiki.openoffice.org/w/index.php?title=Product_Release&oldid=195944, 30.11.2021.

3.2.1.3 bitcoin-draft.pdf

Beskrivelse:	Denne filen er et utkast til det endelige whitepaperet som i dag er publisert på bitcoin.org. Denne filen var i sin tid tilgjengelig på bitcoin.org, og Satoshi Nakamoto refererte til denne filen i en e-post til en kryptografi-e-postliste den 1.november 2018 [9].
Filtype:	Denne filen er et PDF-dokument.
Opprettet:	Fra filens metadata fremgår det at filen ble opprettet 03.10.2008 kl. 13:49:58 lokal tid, i tidssone UTC – 07:00.
Sist endret:	I likhet med «bitcoin.pdf» har ikke denne filen noen dato for sist endret.
Producer:	Filen er generert fra «Writer», med en PDF-versjon 1.4, fra Acrobat 5-serien, i likhet med det originale whitepaperet «bitcoin.pdf».
Creator:	Av metadataene fremgår det at filen ble laget via OpenOffice versjon 2.4.
Andre metadata:	Utover dette inneholder filen svært lite metadata, i likhet med det originale whitepaperet «bitcoin.pdf». Se *Vedlegg 5: Metadata «bitcoin-draft.pdf»* for detaljer.

[9] Hentet fra http://satoshinakamoto.me/2008/11/01/bitcoin-p2p-e-cash-paper/ 03.12.21.

3.2.2 Bilagenes metadata

I denne seksjonen vil vi redegjøre for våre observasjoner til bilagene fra *Vedlegg 1: Filoversikt mottatte filer* som vi har blitt bedt om å analysere. Vi vil presentere observasjonene sekvensielt etter bilagsnummerering. For enkelte av bilagene med like observasjoner har vi valgt å presentere observasjonene gruppert.

Tabell med metadata fra FTK for bilagene:

Fil-navn	MD5-hash	Date created	Date modified	Creator	Producer
Bilag 19.pdf	795a79a8d6526531c9f42fc56aa31665	03.08.2021 12:35:19 (2021-08-03 10:35:19 UTC)	03.08.2021 12:35:20 (2021-08-03 10:35:20 UTC)	Acrobat PDFMaker 17 for Microsoft Outlook	Adobe PDF Library 17.11.238
Bilag 20.PDF	8408769f3720b76e24f54b05716dcbc5	10.09.2019 14:22:59 (2019-09-10 12:22:59 UTC)		Canon iR-ADV C5560 PDF	Adobe PSL 1.3e for Canon
Bilag 21.PDF	d060e9d552fee80da80fb6831ea44ce2	17.09.2019 12:10:22 (2019-09-17 10:10:22 UTC)		Canon iR-ADV C5560 PDF	Adobe PSL 1.3e for Canon
Bilag 22.PDF	71ce1d59067745a639ee4006253539ef	18.09.2019 10:48:15 (2019-09-18 08:48:15 UTC)		Canon iR-ADV C5560 PDF	Adobe PSL 1.3e for Canon
Bilag 23.PDF	75813b22cc54ca604cb703ba9858d509	17.09.2019 13:09:38 (2019-09-17 11:09:38 UTC)		Canon iR-ADV C5560 PDF	Adobe PSL 1.3e for Canon
Bilag 24.PDF	f48e8edcb2c9137ab0595a5bc004fb17	17.09.2019 13:18:44 (2019-09-17 11:18:44 UTC)		Canon iR-ADV C5560 PDF	Adobe PSL 1.3e for Canon
Bilag 25.PDF	ce6335fffb3845f2f9956d16bd3428d0	17.09.2019 14:41:45 (2019-09-17 12:41:45 UTC)		Canon iR-ADV C5560 PDF	Adobe PSL 1.3e for Canon
Bilag 26.PDF	457785a0691cee2915ef34d932d9ce2d	17.09.2019 15:18:54 (2019-09-17 13:18:54 UTC)		Canon iR-ADV C5560 PDF	Adobe PSL 1.3e for Canon
Bilag 27.ODT	7f8befdd723ff197f461f7fba21b32fb				
Bilag 28.PDF	42fa5efcd463e895c4a1aa7f5612f02f	24.03.2009 18:33:15 (2009-03-24 17:33:15 UTC)	21.05.2008 20:43:08 (2008-05-21 18:43:08 UTC)	Writer	OpenOffice.org 2.4
Bilag 29.PDF	ec9a2e200159fb5a06b54cf1ba286647	09.06.2008 12:25:56 (2008-06-09 10:25:56 UTC)	09.06.2008 12:27:02 (2008-06-09 10:27:02 UTC)	PScript5.dll Version 5.2.2	Acrobat Distiller 15.0 (Windows)
Bilag 30.DOC	080a98f16eeeda8defbd15e2b4fac7f2	01.10.2008 14:18:00 (2008-10-01 12:18:00 UTC)	23.10.2008 14:17:00 (2008-10-23 12:17:00 UTC)	Microsoft Office Word	
Bilag 31.DOC	4128db5c270060e1464b9d516b3de8ea	23.10.2008 14:17:00 (2008-10-23 12:17:00 UTC)	23.10.2008 14:19:00 (2008-10-23 12:19:00 UTC)	Microsoft Office Word	

Bilag 32.DOC	b4fe862a3dc51011f1a4bfe0c53159e9	14.11.2008 22:06:00 (2008-11-14 21:06:00 UTC)	16.11.2008 06:18:00 (2008-11-16 05:18:00 UTC)	Microsoft Office Word	
Bilag 33.TIF	5dfa87ac73f3061f3e92a865de7a1f36				
Bilag 34.TIF	49d03aa0f320de2e9711e5a3986195d6				
Bilag 55.PDF	6127f21e473afc118670ffce825f3f50	16.01.2009 02:38:43 (2009-01-16 01:38:43 UTC)	16.01.2009 02:38:43 (2009-01-16 01:38:43 UTC)	Photo Printing Wizard	
Bilag 56.mht	96c18ac5650d67fbfee4c677f071741d				
Bilag 91.PDF	44f48ba4dff491a4faca33a7fa47902b	10.02.2020 12:47:40 (2020-02-10 11:47:40 UTC)		Canon iR-ADV C5560 PDF	Adobe PSL 1.3e for Canon

Vi har i all hovedsak valgt å analysere bilagenes *vedlegg*, da disse skal representere bilagene i sin originalform. Der vi har brukt selve bilaget for våre analyser har vi spesifisert dette ved bruk av bilagets filnavn.

3.2.2.1 Bilag 19

Beskrivelse: «Bilag19.pdf» er fremlagt som en pdf-utskrift av en e-post.

Filtype: Denne filen er en PDF-fil.

Opprettet: Filen er opprettet 03.08.2021 11:35:19 i tidssone UTC +01:00.

Dateringen som indikerer sendt-dato fra innholdet i e-posten stemmer ikke overens med PDF-filens metadata. PDF-filen er laget flere år i ettertid. Uten e-postfilen i originalformat (enten som sikring av epostkontoen som helhet, eller epostmeldingen lagret i epostformat som .MSG) er det umulig å stadfeste om e-posten er sendt eller mottatt på det tidspunktet innholdet antyder. Om man hadde hatt tilgang til epostkontoen i sin helhet eller filen i .MSG-format kunne man verifisere metadata fra e-post header. E-post header inneholder informasjon om forsendelsen, blant annet tidspunkt for forsendelsen, hvilke servere som har vært involvert i forsendelse, avsender og mottaker. Slik det fremstår i Bilag 19, er det kun et dokument med tekst, men innholdet ser ut som en e-post lagret som PDF.

Sist endret: Filen er sist endret 03.08.2021 11:35:20 i tidssone UTC +01:00.

Creator: Denne PDF-filen er produsert ved hjelp av Acrobat PDFMaker 17 for Microsoft Outlook som kan indikere at filen er laget fra Microsoft Outlook.

Producer: Adobe PDF Library 17.11.238

3.2.2.2 Bilag 20, 21 og 23

Beskrivelse: «Bilag 20.pdf», «Bilag 21.pdf» og «Bilag 23.pdf» er scans av papirdokumenter med innhold som er skrevet både digitalt og for hånd.

Filtype: Disse tre filene er PDF-dokumenter.

Opprettet: De tre filene har dato for opprettelse mellom 10.-17.september 2019, i tidssone UTC +1.

Sist endret: Filene har ikke metadata for dato sist endret.

Producer: Adobe PSL 1.3e for Canon.

Creator: Filene er scannet og laget med en Canon imageRunner Advance C5560, Denne scanneren er fra en modell-serie som ble lansert i 2016[10].

Andre observasjoner: Siden innholdet er fra papirdokumenter, er det ikke mulig å tidsbestemme når innholdet i dokumentene ble produsert eller når dokumentene eventuelt har vært skrevet ut.

3.2.2.3 Bilag 22, 24 og 25

Beskrivelse: «Bilag 22.pdf», «Bilag 24.pdf» og «Bilag 25.pdf» er fremlagt som scans av håndskrevne dokumenter.

Filtype: Disse tre filene er tre PDF-filer.

Opprettet: Alle tre filene har en opprettelsesdato i september 2019. I likhet med bilag 20,21 og 23. Siden de er scannede filer av håndskrevne dokumenter er det ikke mulig å fastslå når innholdet er produsert.

Sist endret: Denne datoen finnes ikke.

Creator: I likhet med bilag 20,21 og 23 er har disse tre filene metadata som indikerer at filene ble scannet og laget med en Canon imageRunner Advance C5560 i september 2019. Denne scanneren er fra en modell-serie som ble lansert i 2016[11].

Producer: Adobe PSL 1.3e for Canon.

3.2.2.4 Bilag 26

Beskrivelse: «Bilag 26.pdf» er fremlagt som en scan av en artikkel fra JSTOR.org, som det er gjort håndskrevne notater i.

---

[10] Hentet fra https://www.betterbuys.com/office-equipment/reviews/canon-imagerunner-advance-c5500-series/ 25.11.2021

[11] Hentet fra https://www.betterbuys.com/office-equipment/reviews/canon-imagerunner-advance-c5500-series/, 30.11.2021.

Filtype:	Denne filen er en PDF-fil.
Opprettet:	Filen har en opprettelsesdato i september 2019, i likhet med bilag 20, 21, 22, 23, 24 og 25.
Dato sist endret:	Eksisterer ikke.
Creator:	Filen er i likhet med bilag 20, 21, 22, 23, 24 og 25, scannet og laget med Canon imageRunner Advance C5500 Series.
Producer:	Adobe PSL 1.3e for Canon.

3.2.2.5 Bilag 27

Beskrivelse:	«Bilag 27.odt» fremstår å være en versjon av et whitepaper.
Filtype:	Denne filen er et Open Office-dokument.
Opprettet:	«Bilag 27.odt» har manglende metadata for «Date created» på filnivå. Dette er ikke forventet, og selv om man resetter metadata-datoene i Open Office-applikasjonen som KPMG har gjort på et testdokument i *Vedlegg 9: Metadata «ODT Testdokument.odt»*, vil man fortsatt forvente å finne en dato for opprettelse. Dette kan tyde på at denne har blitt overskrevet eller slettet.
Sist endret:	«Bilag 27.odt» har manglende metadata for «Date modified» på filnivå. Dette er ikke forventet, med mindre man resetter metadata-datoene i Open Office-applikasjonen som KPMG har gjort på et testdokument i *Vedlegg 9: Metadata «ODT Testdokument.odt».*
Producer:	«Bilag 27.odt» har manglende metadata for «File producer». Dette er forventet.
Creator:	«Bilag 27.odt» har manglende metadata for «File creator». Dette er forventet.
Andre metadata:	Det er lite metadata tilgjengelig for denne filen.

3.2.2.6 Bilag 28

Beskrivelse:	«Bilag 28.pdf» fremstår å være en versjon av et whitepaper.
Filtype:	Denne filen er et PDF-dokument.
Opprettet:	«Bilag 28.pdf» har mange samsvarende metadata med filen som er publisert på SSRN.com. Ut fra filens metadata ble filen opprettet 24.03.2009 kl. 11:33:15 lokal tid, i tidssone UTC – 06:00. Dette samsvarer med opprettelsesdato for «bitcoin.pdf», det originale whitepaperet som er offentlig tilgjengelig for nedlasting på bitcoin.org. Ut fra XMP-metadataene knyttet til filen, er filen opprettet 24.01.2008 kl. 11:33:15 lokal tid, i tidssone UTC - 06:00. Det er derfor ikke samsvar mellom filens metadata og XMP-

	metadataene, som kan være en indikasjon på at XMP-metadataene er endret. Det er også verdt å merke seg at selv om opprettelsesdatoen i XMP-metadataene er over ett år tidligere enn filens opprettelsesdato, er tidsstemplingen nøyaktig den samme, i samme tidssone UTC- 06:00.
Sist endret:	Filens metadata for sist endring er 21.05.2008 kl. 18:43:08 lokal tid, i tidssone UTC +01:00. Dette samsvarer med filens XMP-metadata. Begge disse tidsstemplene er i tidssone UTC +01:00, som er en annen tidssone enn filens opprettelsesdato.
Producer:	PDF-filen ser ut til å ha blitt laget via OpenOffice versjon 2.4.
Creator:	Filen er generert fra «Writer», og har i likhet med SSRN.com-versjonen en PDF-versjon 1.6, fra Acrobat 7-serien, som er en senere versjon enn det originale Bitcoin-whitepaperet har.
Andre metadata:	I likhet med «SSRN-id3440802.pdf» har «Bilag 28.pdf» mer metadata enn det «bitcoin.pdf» innehar, som eksempelvis keywords, author, title med mer. Se *Vedlegg 10: Metadata «Bilag 28.pdf»* for detaljer. Dette er metadata som man kan legge inn direkte ved hjelp av en programvare som eksempelvis Adobe Acrobat Pro. Vi observerer også at filen har XMP-metadata attributter fra Photoshop, til forskjell fra «bitcoin.pdf».

3.2.2.7 Bilag 29

Beskrivelse:	«Bilag 29.pdf» er et flytskjema.
Filtype:	«Bilag 29.pdf» er en PDF-fil.
Opprettet:	Fra filens metadata fremstår det som om opprettelsesdato er 09.06.2008, med tidsstempling kl. 11:25:56 i lokal tid, tidssone +01:00. Nede til høyre i filen inneholder den tekst med en datotidsstempling «9/06/2008 11:24». Denne avviker fra tidspunktet i filens opprettelsesdato. Det er også usikkert hvilket datoformat denne datoen benytter seg av.
Sist endret:	09.06.2008, 11:27:02 i lokal tid, tidssone +01:00.
Creator:	PScript5.dll Version 5.2.2
Producer:	PDF-filen er laget med Adobe Acrobat Distiller 15.0 for Windows, som er en programvare for å konvertere PostScript-filer til PDF. Adobe Acrobat Distiller 15.0 ble lansert i 7.april 2015 [12]. Det er derfor usannsynlig at denne filen kan ha vært laget eller sist endret 09.06.2008 som metadataene antyder. Metadataene virker derfor å være endret.

[12] Hentet fra https://web.archive.org/web/20150415142616/https://itunes.apple.com/us/app/adobe-acrobat-dc-pdf-reader/id469337564?mt=8, 07.12.21

3.2.2.8 Bilag 30

Beskrivelse: «Bilag 30.doc» er et tekstdokument med det som ser ut til å være notater.

Filtype: «Bilag 30.doc» er et Microsoft Office Word 97-2003 dokument.

Opprettet: Fra filens metadata er filen **Opprettet** 01.10.2008 kl. 23:18:00 lokal tid.

Sist endret: Fra filens metadata er filen **Sist endret** 23.10.2008 kl. 23:17:00 lokal tid.

Andre observasjoner: Differanse mellom Metadata-tidspunktene fra Word for "**Opprettet"** og "**Sist endret"** er 21 dager, 23 timer og 53 minutter. Dette er tilnærmet lik (sekunder er ikke medregnet og kan forklare avviket) "**Samlet Redigeringstid"** som angis i Word til **31676** minutter (21 dager, 23 timer og 56 minutter), og vil i så fall bety at filen har stått åpent i Word-applikasjonen i nær 22 dager. Se *Vedlegg 12: Metadata «Bilag 30.doc»* for detaljer.

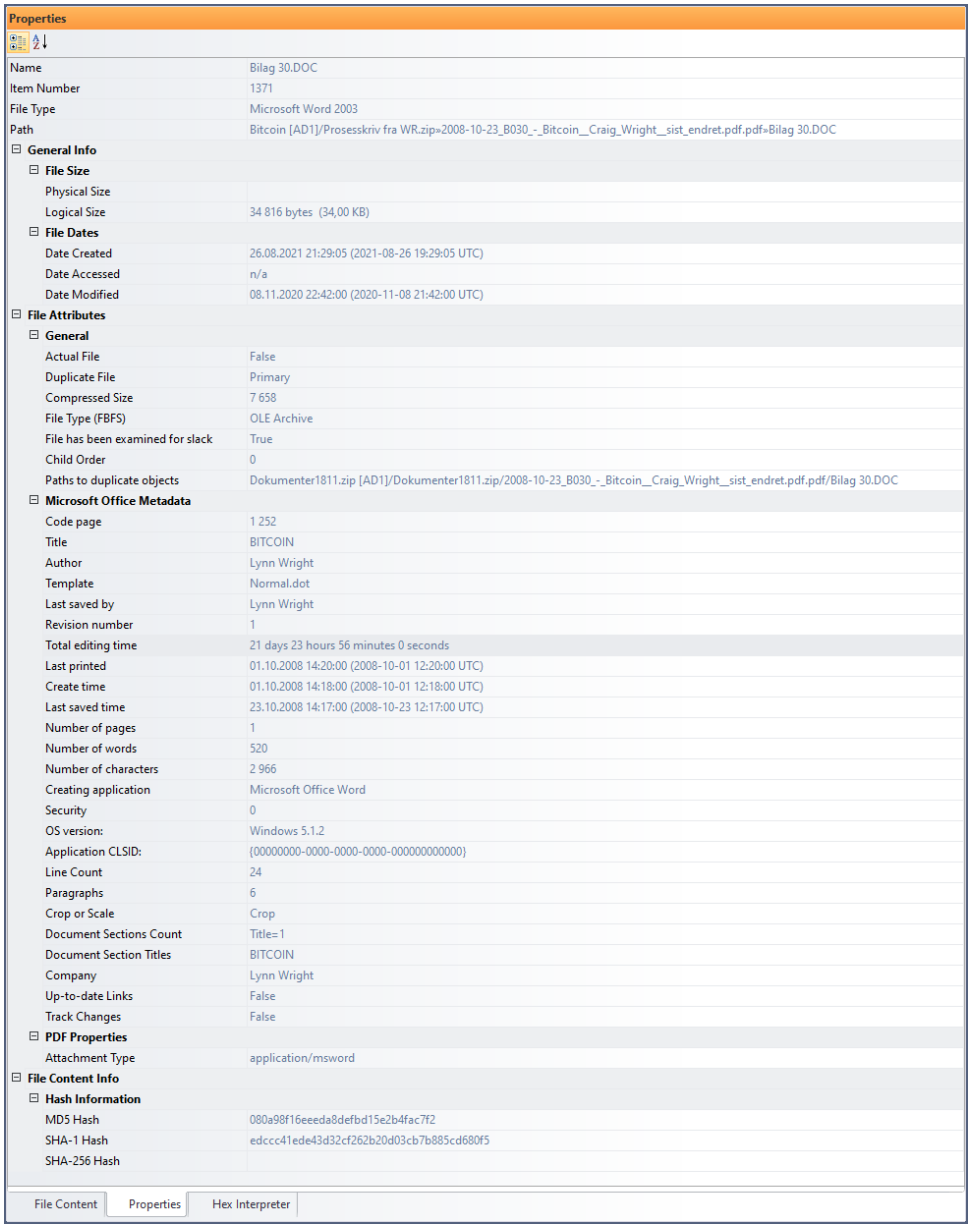
FTK rapporterer «Total editing time» til 21 days 23 hours 56 minutes 0 seconds som stemmer overens med differansen "Opprettet" og "Sist endret" i Word. FTK rapporterer «Total editing time» til «21 days 23 hours 56 minutes 0 seconds» som stemmer overens med differansen "Opprettet" og "Sist endret" i Word. Dette indikerer at klokke kan være manipulert fordi ved et normaltilfelle hvor et dokument står åpent og endres/lagres fortløpende vil FTK rapportere «Total Editing Time» som reflekterer faktisk redigeringstid, ikke kun differanse mellom «Sist Endret» og «Opprettet» som MS Word oppgir. FTK viser at revisjonsnummeret er «1» som betyr at dokumentinnhold er lagret kun én gang.

Utført analyse sett opp mot testing av «Total Editing Time» gjør at vi vurderer filens metadata som usikre.

Andre Observasjoner: Bilag 30.doc er lagret i Word 97-2003 formatet .**doc**. Dokumentet er laget ut fra malen **Normal.dotm**.

Filformatet **docx** (dokument) og **dotm** (dokumentmal) ble lansert til Word 2007, mens **doc** (dokument) og **dot** (dokumentmal) er det gamle filformatet til Microsoft Word.

Informasjonen om «Normal.dotm» som mal vises både i MS Word og i FTK:

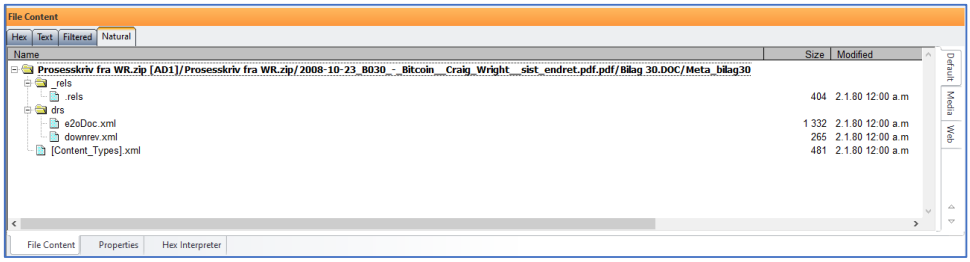


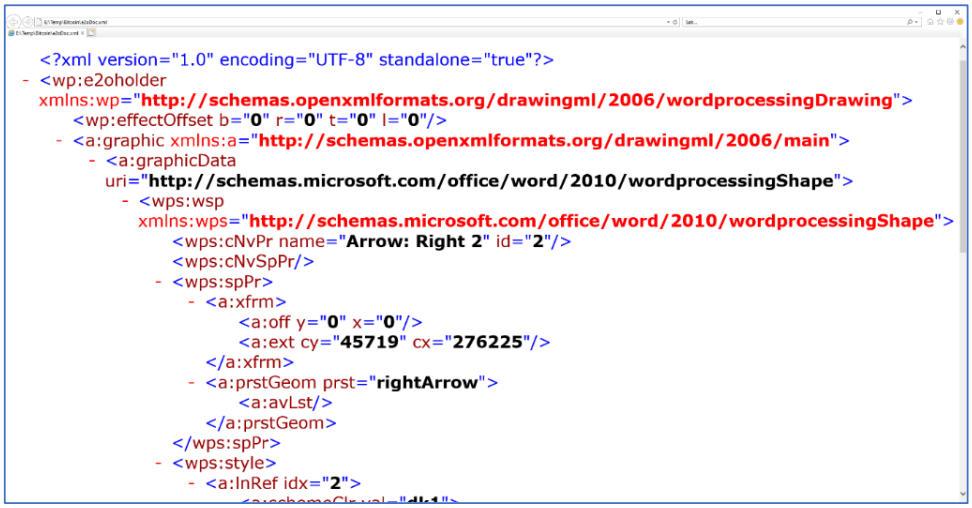
Properties

Name	Bilag 30.DOC
Item Number	1371
File Type	Microsoft Word 2003
Path	Bitcoin [AD1]/Prosesskriv fra WR.zip»2008-10-23_B030_-_Bitcoin__Craig_Wright__sist_endret.pdf.pdf»Bilag 30.DOC
General Info	
File Size	
Physical Size	
Logical Size	34 816 bytes (34,00 KB)
File Dates	
Date Created	26.08.2021 21:29:05 (2021-08-26 19:29:05 UTC)
Date Accessed	n/a
Date Modified	08.11.2020 22:42:00 (2020-11-08 21:42:00 UTC)
File Attributes	
General	
Actual File	False
Duplicate File	Primary
Compressed Size	7 658
File Type (FBFS)	OLE Archive
File has been examined for slack	True
Child Order	0
Paths to duplicate objects	Dokumenter1811.zip [AD1]/Dokumenter1811.zip/2008-10-23_B030_-_Bitcoin__Craig_Wright__sist_endret.pdf.pdf/Bilag 30.DOC
Microsoft Office Metadata	
Code page	1 252
Title	BITCOIN
Author	Lynn Wright
Template	Normal.dot
Last saved by	Lynn Wright
Revision number	1
Total editing time	21 days 23 hours 56 minutes 0 seconds
Last printed	01.10.2008 14:20:00 (2008-10-01 12:20:00 UTC)
Create time	01.10.2008 14:18:00 (2008-10-01 12:18:00 UTC)
Last saved time	23.10.2008 14:17:00 (2008-10-23 12:17:00 UTC)
Number of pages	1
Number of words	520
Number of characters	2 966
Creating application	Microsoft Office Word
Security	0
OS version:	Windows 5.1.2
Application CLSID:	{00000000-0000-0000-0000-000000000000}
Line Count	24
Paragraphs	6
Crop or Scale	Crop
Document Sections Count	Title=1
Document Section Titles	BITCOIN
Company	Lynn Wright
Up-to-date Links	False
Track Changes	False
PDF Properties	
Attachment Type	application/msword
File Content Info	
Hash Information	
MD5 Hash	080a98f16eeeda8defbd15e2b4fac7f2
SHA-1 Hash	edccc41ede43d32cf262b20d03cb7b885cd680f5
SHA-256 Hash	

File Content | Properties | Hex Interpreter

Vi har benyttet FTK for å trekke ut metadata fra Bilag 30.doc i separate filer. I en av disse filene, e20Doc.xml, finnes referanse til http://schemas.microsoft.com/office/word/**2010**/wordprocessingShape, som kan tyde på at filen er laget i 2010 eller senere.





3.2.2.9 Bilag 31

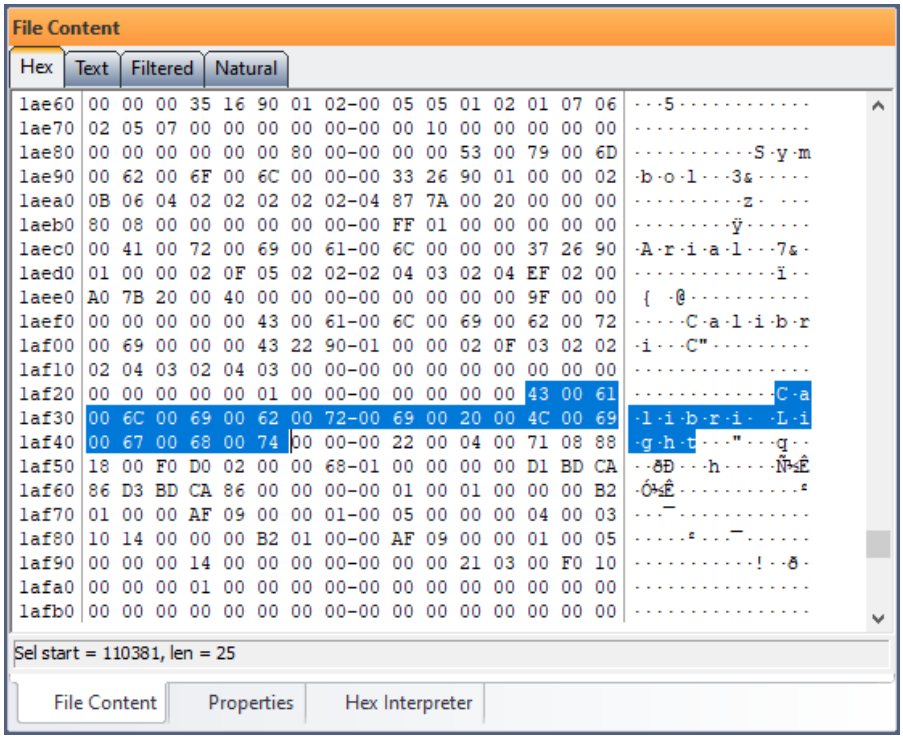
Beskrivelse: «Bilag 31.doc» er et tekstdokument med det som ser ut til å være notater.

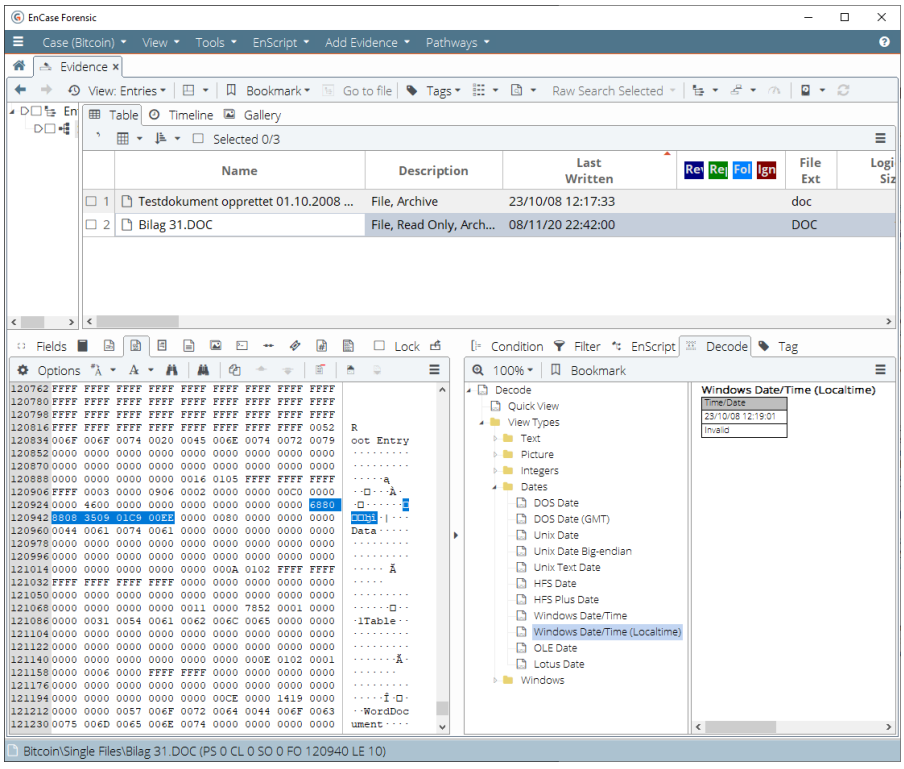
Filtype: «Bilag 31.doc» er et Microsoft Office Word 97-2003 dokument.

Opprettet: Fra filens metadata er filen opprettet 23.10.2008 kl. 23:17:00 lokal tid, i tidssone UTC +11, som blant annet inkluderer Australian Eastern Daylight Time. Vi observerer at opprettelsesdato for «Bilag 31.DOC» sammenfaller med «Date modified» for «Bilag 30.DOC».

Sist endret: Fra filens metadata er filen sist endret 23.10.2008 kl. 23:19:00 lokal tid, i tidssone UTC +11, som blant annet inkluderer Australian Eastern Daylight Time.

Andre observasjoner: Fra skjermbildet fra FTK under ser vi at filen inneholder fonten «Calibri Light»:





Fra EnCase-skjermbildet over ser vi at verdien for «Root Entry» i «Bilag 31.doc» viser tidsstempel 23.10.2008 12:18 (Lokal tid, 14:19 UTC) som indikerer tidspunkt for når filen sist ble oppdatert.

Calibri Light ble først lansert som en del av Calibri-fontfamilien i 2.00-versjonen med en copyright fra 2012 som en del av Windows 8 lanseringen [13] [14]. Fonten ble ved lansering i Windows 8 også lansert for Windows 7 og Windows Server 2008 R2.

Fra fontlistene til Office 2007 og Office 2010, så finnes ikke Calibri Light i disse:

Skjermbilde med fontlister til Office 2007 fra web-arkivet til Microsoft Support [15]

BSSYM7.TTF	Bookshelf Symbol 7 (TrueType)
CALIBRI.TTF	Calibri (TrueType)
CALIBRIB.TTF	Calibri Bold (TrueType)
CALIBRII.TTF	Calibri Italic (TrueType)
CALIBRIZ.TTF	Calibri Bold Italic (TrueType)
CALIFB.TTF	Californian FB Bold (TrueType)
CALIFI.TTF	Californian FB Italic (TrueType)

Skjermbilde med fontlister til Office 2010 fra web-arkivet til Microsoft Support [16]

BSSYM7.TTF	Bookshelf Symbol 7 (TrueType)
CALIBRI.TTF	Calibri (TrueType)
CALIBRIB.TTF	Calibri Bold (TrueType)
CALIBRII.TTF	Calibri Italic (TrueType)
CALIBRIZ.TTF	Calibri Bold Italic (TrueType)
CALIFB.TTF	Californian FB Bold (TrueType)
CALIFI.TTF	Californian FB Italic (TrueType)

Men fonten finnes i font-listen for Office 2013 produkter:

Skjermbilde med fontlister til Office 2013 fra Microsoft Support [17]

[13] Hentet fra https://web.archive.org/web/20121104171948/http://www.microsoft.com/typography/fonts/font.aspx?FMID=1917, 09.12.21.

[14] Hentet fra https://support.microsoft.com/en-us/topic/an-update-is-available-to-add-the-calibri-light-and-calibri-light-italic-fonts-to-windows-7-and-windows-server-2008-r2-717a0bbe-0610-bd3c-3cc1-bad9b7809e55, 09.12.21

[15] Hentet fra https://web.archive.org/web/20110221084933/http://support.microsoft.com/kb/924623, 10.12.21

[16] Hentet fra https://web.archive.org/web/20100625153814/http://support.microsoft.com/kb/2121313, 10.12.21

[17] Hentet fra https://support.microsoft.com/en-us/topic/35dfa558-f367-43e6-8f62-7305096b5c21, 10.12.21

More Information

The following table lists the fonts that are installed by Microsoft Office Professional Plus 2013:

Note: Other versions of Office 2013 install a smaller selection of fonts.

File	Font name
CalibriL.ttf	Calibri Light
CalibriLI.ttf	Calibri Light Italic

Windows 8 ble lansert i 2012 og basert på dette må derfor Bilag 31.doc være opprettet i 2012 eller senere. På dette grunnlaget virker metadata for datoene for **Opprettelse** og **Sist endret** å være manipulert.

3.2.2.10 Bilag 32

Beskrivelse: Ut fra innhold fremstår «Bilag 32.doc» å være en versjon av et Bitcoin-whitepaper.

Filtype: «Bilag 32.doc» er et Microsoft Word 97-2003 dokument.

Opprettet: Fra filens metadata fremgår det at filen ble opprettet 14.11.2008 21:06 UTC, eller 15.11.2008 kl. 08:06:00 lokal tid, som tilsvarer tidssone UTC + 11:00, som blant annet inkluderer Australian Eastern Daylight Time.

Sist endret: Fra filens metadata fremgår det at filen ble sist endret 16.11.2008 kl. 05:18 UTC, eller 16.11.2008 kl. 16:18:00 lokal tid, som vil tilsvare tidssone UTC +11:00, som blant annet inkluderer Australian Eastern Daylight Time.

Producer: Ingen informasjon, som er normalt for MS Word dokumenter.

Creator: Microsoft Office.

Andre metadata: «Bilag 32.doc» innehar mer metadata, som eksempelvis author, title med mer, som er tilgjengelig i *Vedlegg 14: Metadata «Bilag 32.doc»*.

FTK rapporterer filens «Total editing time» til 1 days 8 hours 1 minutes 0 seconds som stemmer overens med differansen mellom datoene for «Opprettet/Created» og «Sist Skrevet Ut/Last Printed» som vi finner i Word. Dette indikerer at klokke er manipulert da man i et normaltilfelle hvor et dokument står åpent og endres/lagres fortløpende vil FTK rapportere «Total Editing Time» som reflekterer faktisk redigeringstid, ikke kun differanse mellom «Sist Endret» og «Opprettet» som MS Word oppgir. Se *Vedlegg 14: Metadata «Bilag 32.doc»* for mer detaljer rundt dette.

3.2.2.11 Bilag 33

Beskrivelse: «Bilag 33.TIF» er et flytskjema. Filformatet TIF/TIFF (Tagged Image File Format) er et filformat ofte benyttet av Adobe, skanner-applikasjoner eller Fax.

Metadata: En TIF eller TIFF-fil inneholder sjeldent mye metadata, og dette er også tilfellet her.

3.2.2.12 Bilag 34

Beskrivelse: «Bilag 34.TIF» er et flytskjema. Filformatet TIF/TIFF (Tagged Image File Format) er et filformat ofte benyttet av Adobe, skanner-applikasjoner eller Fax.

Metadata: En TIF eller TIFF-fil inneholder sjeldent mye metadata, og dette er også tilfellet her.

3.2.2.13 Bilag 35 – 48

Disse bilagene er kategorisert som *Kildekode*. For observasjoner av disse, se seksjon **3.3 Kildekode**.

3.2.2.14 Bilag 49 og 74

Beskrivelse: Bilagene «Bilag 49.ICO» og «Bilag 74.ICO» er to bildefiler som viser Bitcoin ikonet. Begge bilag inneholder nøyaktig samme fil.

Observasjon: Det er lite metadata knyttet til disse filene.

3.2.2.15 Bilag 50 og 57

Beskrivelse: «Bilag 50.exe» og «Bilag 57.exe» er to eksekverbare filer (Programfiler).

Observasjon: Det er lite metadata fra FTK knyttet til disse .exe-filene, men viktige observasjoner rundt filenes innhold som er spesifisert i seksjon ***3.2.4.16***.

3.2.2.16 Bilag 51 – 53

Disse bilagene er kategorisert som *Kildekode*. For observasjoner av disse, se seksjon **3.3 Kildekode.**

3.2.2.17 Bilag 54 og 90

Beskrivelse: «Bilag 54.LOG» og «Bilag 90.LOG» er to loggfiler.

Metadata: Det finnes ingen tidsstempler i disse filene.

3.2.2.18 Bilag 55

Beskrivelse: «Bilag 55.pdf» er et Memo-dokument basert på en Microsoft Word-mal lagret som FAX og deretter skrevet ut til en PDF-fil.

Filtype: Denne filen er en PDF-fil.

Opprettet: Filen er opprettet 16.01.2009 16:01:38 UTC

Sist endret: Filen er sist endret 16.01.2009 16:01:38 UTC

Creator: Photo Printing Wizard

Author: Lynn Wright

Andre observasjoner: Ut fra Microsoft Office Metadata, er det KPMGs vurdering at dette en PDF fil basert på et .FAX-dokument som igjen er basert på et Microsoft mal dokument[18]. FAX-filer kan leses med Microsoft Photo Viewer.

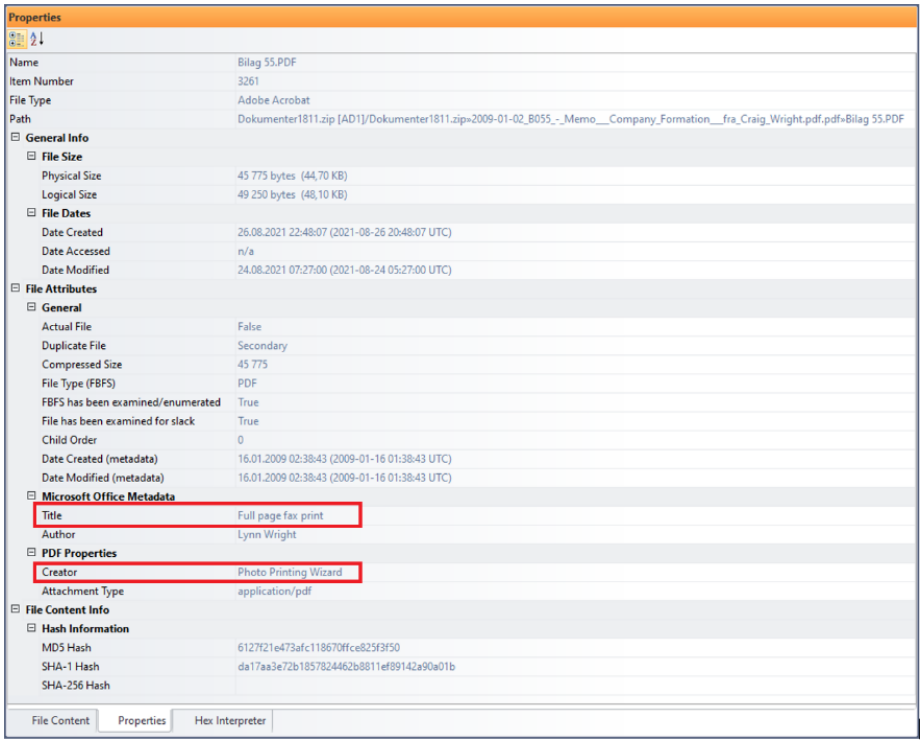
Basert på navngivningen (Title = "**Full page fax print**") er det sannsynlig at brukeren har valgt «Print» funksjonen i Windows Utforsker og deretter skrevet ut til PDF. På samme måte som om man velger å skrive ut et bilde fra Windows Utforsker hvor «Title» vil bli «Full Page Photo»

Når en Word-mal åpnes, opprettes et nytt dokument som gis et standard navn og løpenummer f.eks. "Microsoft Word – Dokument 1" inntil brukeren lagrer dokumentet med et nytt filnavn.

Ut fra metadata er dokumentet er opprettet 16.02.2009 16:01:38 og skrevet ut på samme tidspunkt som igjen støtter at dette er en utskrift fra Windows Utforsker. Fra dokumentets innhold ser vi at dokumentet er datert 02 Jan 2009:

[18] Hentet fra Interoffice Memo (Professional design), 06.12.21.

«Bilag 55.pdf» FTK Metadata:



3.2.2.19 Bilag 56

Beskrivelse: «Bilag 56.mht» er en e-post.

Filtype: Denne filen er en MHT-fil.

Opprettet: Filen er opprettet 26.08.2021 21:04 UTC

Sist endret: Filen er sist endret 26.08.2021 07:57 UTC

Andre observasjoner: Innholdet i filen er en e-postmelding med vedlegg. Fra innholdet i e-posten fremgår det at e-postmeldingen er sendt «October 11, 2014 4:50 AM».

e-postfilen inneholder 9 ulike vedlegg som alle er skjermbilder i PNG format.

MHT er i hovedsak benyttet av Microsoft som et filformat for lagring av e-post som en web-side som et arkiv og som inkluderer innebygd informasjon som for eksempel vedlegg.

Tidsstemplene viser filens tidsstempler på filsystemet, filformatet har ikke metadata som viser tidsstempler. At tidsstempel for «Opprettet» er tidsmessig etter «Sist endret» viser at denne filen er opprettet på en annen lagringsenhet enn den den ble endret på.

3.2.2.20 Bilag 58 – 73, 75 – 89

Disse bilagene er kategorisert som *Kildekode*. For observasjoner av disse, se seksjon **3.3 Kildekode**.

3.2.2.21 Bilag 91

Beskrivelse:	«Bilag 91.PDF» er en scan av et håndskrevet notat.
Filtype:	Denne filen er en PDF-fil.
Opprettet:	Filen er opprettet 10.02.2020 12:47:40, i tidssone UTC +01:00.
Dato sist endret:	Eksisterer ikke.
Creator:	Filen er i likhet med bilag 20, 21, 22, 23, 24 og 25, scannet og laget med Canon imageRunner Advance C5560. Denne scanneren er fra en modell-serie som ble lansert i 2016 [19]
Producer:	Adobe PSL 1.3e for Canon.

[19] Hentet fra https://www.betterbuys.com/office-equipment/reviews/canon-imagerunner-advance-c5500-series/, 30.11.2021.

3.2.3 Innholds- og formateringsanalyse av referansefiler

I denne seksjonen vil vi presentere våre observasjoner fra analyser av fonter, formattering og innhold for de tre referansefilene. Disse observasjonene vil bli brukt som referanse ved analyser av fonter, formattering og innhold av enkelte av bilagene som er fremlagt.

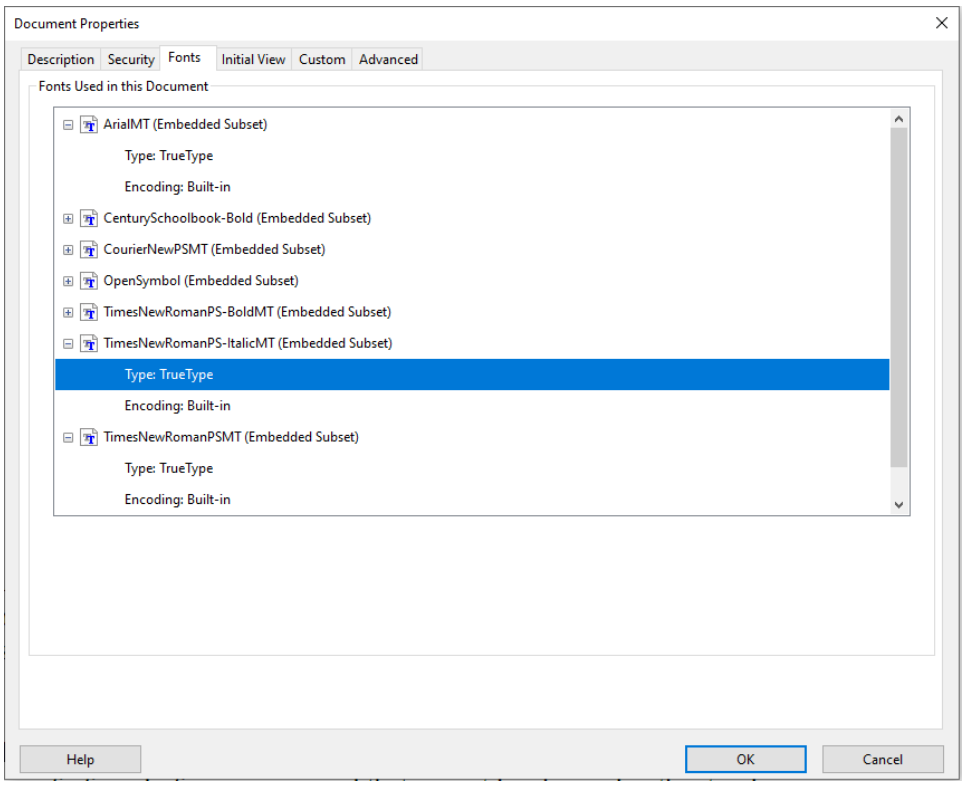
3.2.3.1 Bitcoin.pdf

Beskrivelse: Dette er et whitepaper tilgjengelig fra bitcoin.org [20].

Papirstørrelse: Denne filen bruker «Standard Letter»-størrelse (8,5 x 11 tommer), som er en standardstørrelse i Nord-Amerika [21].

Fonter: Fra Adobe Acrobat observerer vi at filen bruker i hovedsak fonter fra Century Schoolbook-fontfamilien for overskrifter, Times New Roman-fontfamilien for brødtekst, Arial-fontfamilien for tekst figurer, OpenSymbol for formler og Courier New for seksjonen for «c-kode» på side 7 og 8. For fullstendig font-oversikt se *Vedlegg 15: Innholdsanalyse «bitcoin.pdf»*.

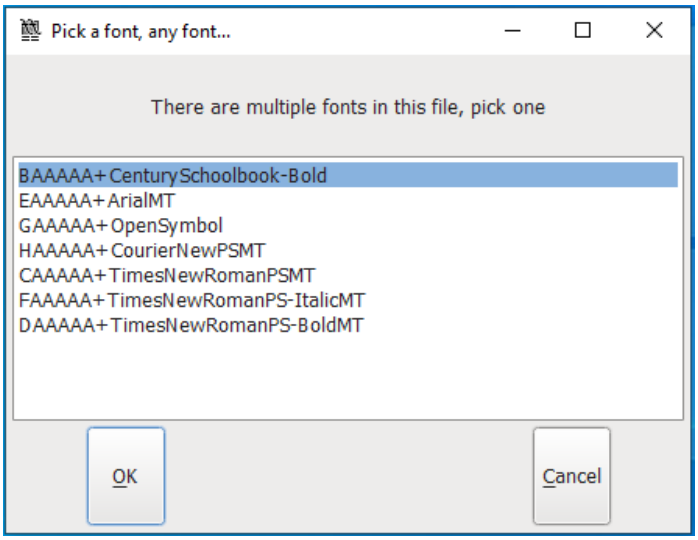
Adobe font-oversikt «bitcoin.pdf»:



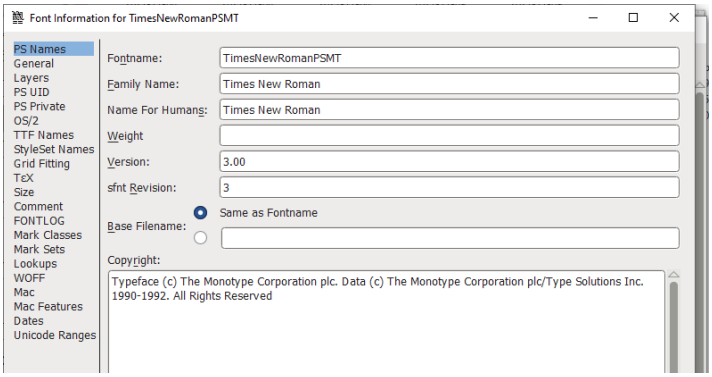
[20] Lastet ned fra https://bitcoin.org/en/bitcoin-paper, 11.11.2021

[21] Hentet fra artikkel fra https://www.swiftpublisher.com/useful-articles/paper-sizes-and-formats-explained 01.12.2012.

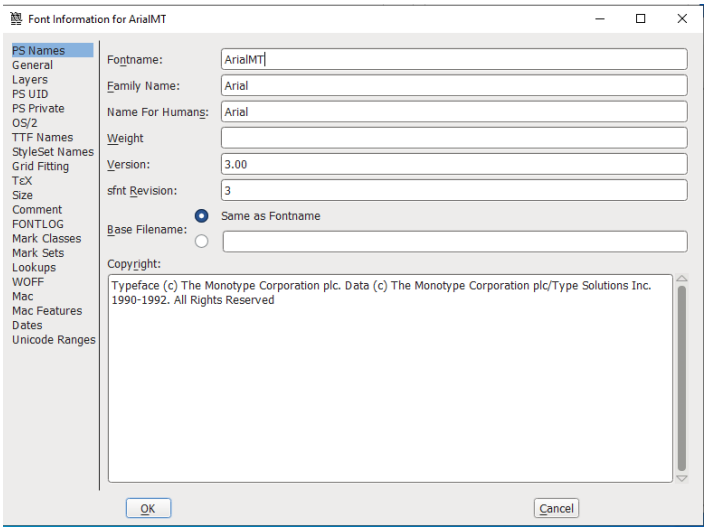
Alle fontene i denne filen bruker *Type: TrueType* og *Encoding: Built-in*. Ved å analysere de innebygde fontene i «bitcoin.pdf» i programmet FontForge kan vi inspisere fontenes metadata. Fra FontForge observerer vi de samme innebygde fontene som fra Adobe Acrobat:

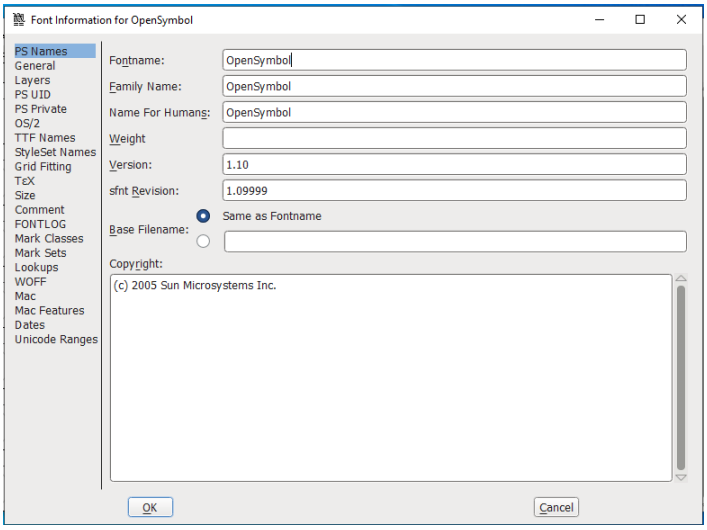


Inspiserer man fonten TimesNewRomanPSMT nærmere observerer vi at denne fonten er innebygget som versjon 3.00 med en copyright fra 1990-1992.



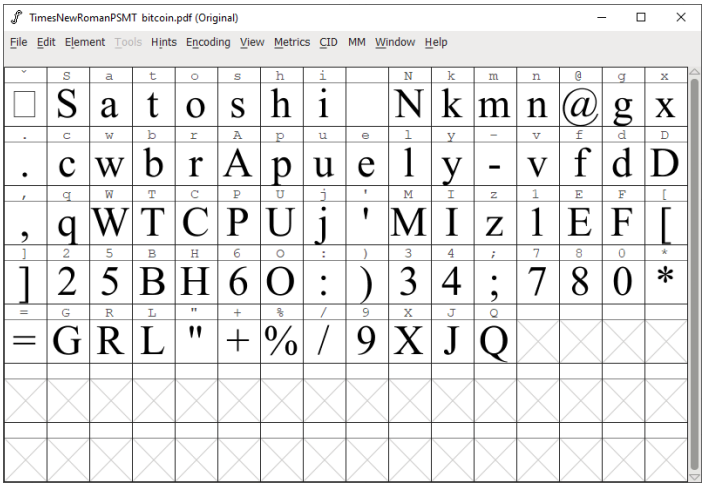
For fontene ArialMT og OpenSymbol observerer vi følgende font-metadata, med henholdsvis copyright fra 1990-1992 og 2005:





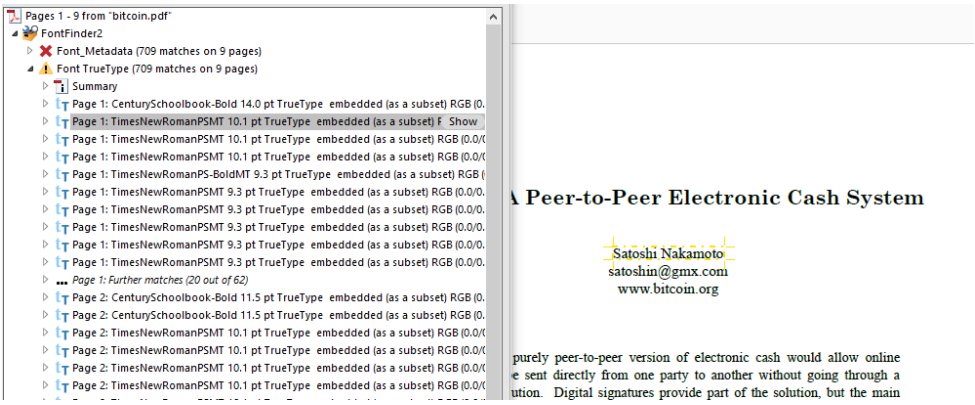
For alle fontene som er brukt i «bitcoin.pdf» finner man copyright-informasjon som stammer fra før filens opprettelses-dato, som forventet.

Inspiserer man fontenes innbygging i filen ytterligere kan man se *hvordan* hver font i filen er komprimert og bygget inn. En subset-innebygget font, som denne filen har, vil bygges inn i dokumentet ved at den komprimerer ned alle tegn som er brukt i filen til en fullstendig tegnoversikt per font. Fra «bitcoin.pdf» ser man eksempelvis hvordan fonten TimesNewRomanPSMT er komprimert ned til følgende tegnoversikt, som angir alle tegn i denne fonten fra filen:

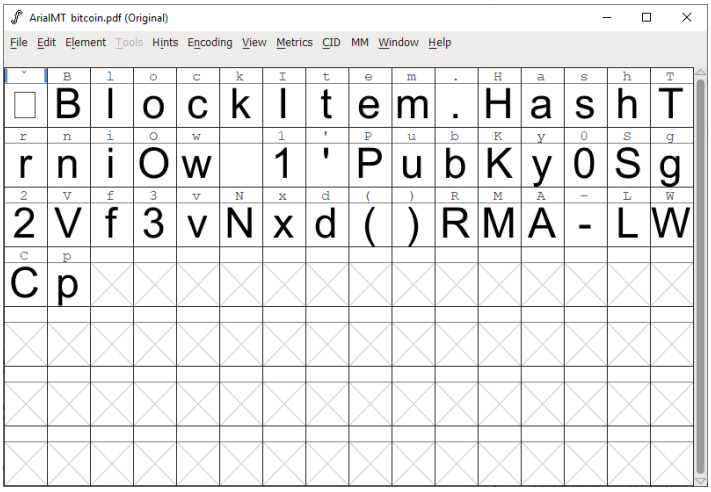


Ser man på de første tegnene skrevet i TimesNewRomanPSMT i «bitcoin.pdf» er dette «Satoshi Nakamoto», slik at «Satoshi» er de første tegnene som blir komprimert til oversikten over. Videre finner man bokstavene «Nkm», som er den komprimerte måten å skrive «Nakamoto» på, siden «a», «o» og «t» allerede er definert tidligere i filen.

Skjermbilde fra «bitcoin.pdf» og første linje med tekst i font TimesNewRomanPSMT:



På lik linje finner man også følgende tegnoversikt for fonten ArialMT:



Disse tegnene representerer alle tegn for fonten ArialMT i «bitcoin.pdf».

For ytterligere informasjon om de resterende fontene brukt i filen og deres metadata se *Vedlegg 15: Innholdsanalyse «bitcoin.pdf».*

For å verifisere resultatet i denne fremgangsmåten for å analysere font-metadata, har KPMG også brukt verktøyet MuPDF for å trekke ut alle font-filene direkte fra «bitcoin.pdf». Deretter har vi analysert metadataene i fontfilene. Resultatet fra de to fremgangsmåtene er identiske. Se *Vedlegg 4.15.4* for skjermbilder på denne fremgangsmåten.

3.2.3.2 SSRN-id3440802.pdf

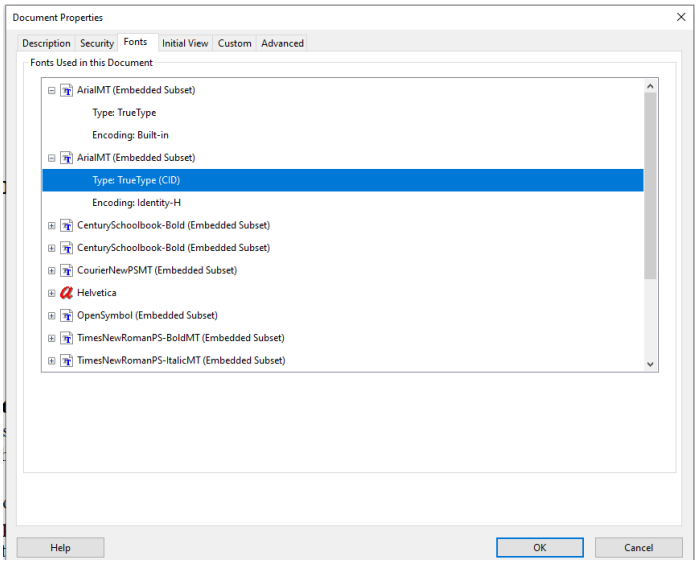
Beskrivelse: Dette er et whitepaper som er publisert på SSRN.com med Craig Wrights navn den 22.august 2019 [22]. Fra SSRN.com står det at filen ble skrevet 21. august 2008.

Papirstørrelse: Denne filen bruker «Standard Letter»-størrelse (8,5 x 11 tommer), i likhet med «bitcoin.pdf».

Fonter: Filen bruker i likhet med «bitcoin.pdf» hovedsak fonter fra Century Schoolbook-fontfamilien for overskrifter, Times New Roman-fontfamilien for brødtekst, Arial-fontfamilien for tekst i figurer, OpenSymbol for formler og Courier New for seksjonen for «c-kode» på side 7 og 8. For en fullstendig oversikt over fonter brukt i denne filen se *Vedlegg 16: Innholdsanalyse «SSRN-id3440802.pdf»* og *Vedlegg 27: Font-oversikt fra tekst: Referansefiler.*

En vesentlig forskjell fra «bitcoin.pdf» er at denne filen bruker både fonter i *Type: TrueType* med *Encoding: Built-in* og *Type: TrueType (CID)* med *Encoding: Identity-H*, som eksempelet under viser:

[22] Lastet ned fra https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3440802, 11.11.2021.



Dette er en enkoding som har blitt mer vanlig i nyere Windows-operativsystemer, da systemfontene har blitt mer komplekse fra Windows 7 til Windows 8 og senere Windows 10 [23]. Windows 7 ble først lansert 22.10.2009 [24]. Windows 8 kom først i oktober 2012 [25].

For å kunne analysere disse forskjellene i font-type og enkoding i FontForge, har KPMG dekomprimert filen «SSRN-id3440802.pdf»s datastrømmer i verktøyet QPDF:

```
$ ../qpdf-10.4.0/bin/qpdf --stream-data=uncompress SSRN-id3440802.pdf SSRN_unc.pdf
```

Deretter har KPMG inspisert den dekomprimerte filens font-datastrømmer (Referansefil «SSRN_unc.pdf» i *Vedlegg 2: Filoversikt referansefiler*) i FontForge. For å verifisere denne fremgangsmåten har KPMG også brukt verktøyet MuPDF [26] for å trekke ut alle font-filene direkte fra «SSRN-id3440802.pdf», for deretter å analysere metadataene i disse fontfilene. Resultatet fra de to fremgangsmåtene er identiske. Se *Vedlegg 4.16.3* for skjermbilder.

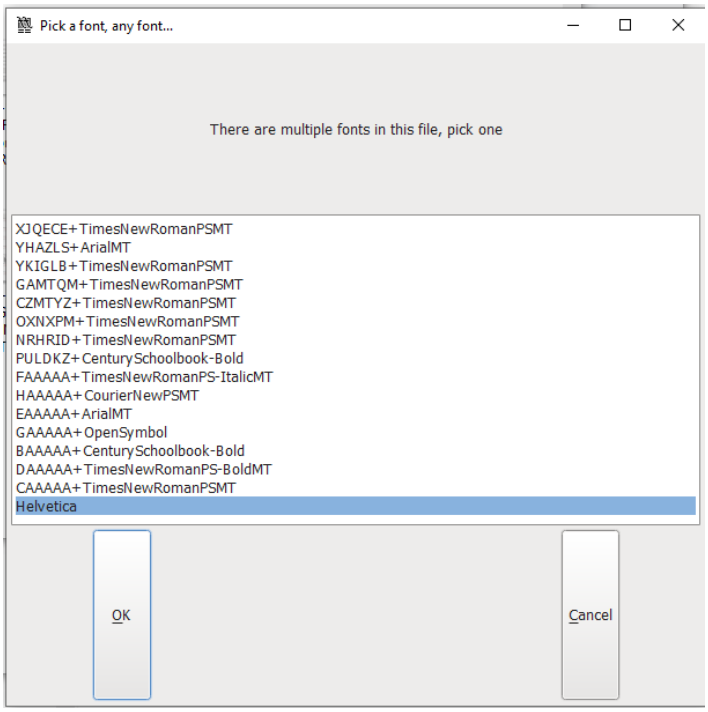
Skjermbildet under viser alle fonter innebygget i filen «SSRN-id3440802.pdf» i FontForge:

[23] Hentet fra https://community.adobe.com/t5/acrobat-discussions/font-encoding-settings-removing-identity-h-encoding/td-p/10605220 01.12.21.

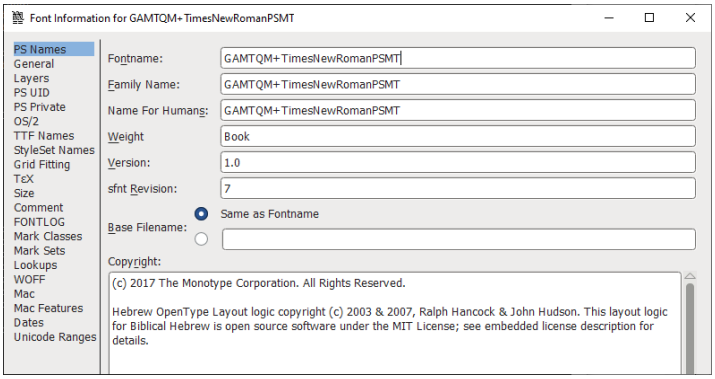
[24] Hentet fra https://blogs.windows.com/windowsexperience/2009/10/22/windows-7-arrives-today-with-new-offers-new-pcs-and-more/, 07.12.2021

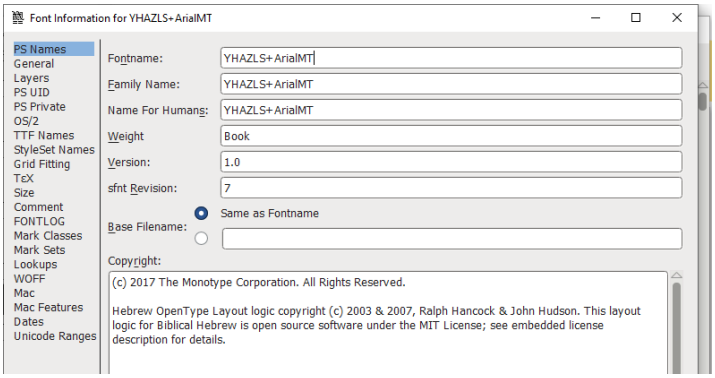
[25] Hentet fra https://web.archive.org/web/20121027202517/https://blogs.windows.com/windows/b/bloggingwindows/archive/2012/10/25/windows-reimagined-windows8.aspx, 01.12.2021

[26] Hentet fra https://mupdf.com/index.html, 10.12.21.



For fontene GAMTQM+ TimesNewRomanPSMT og YHAZLS+ ArialMT ser vi at disse er bygget inn i filen med en copyright fra 2017:

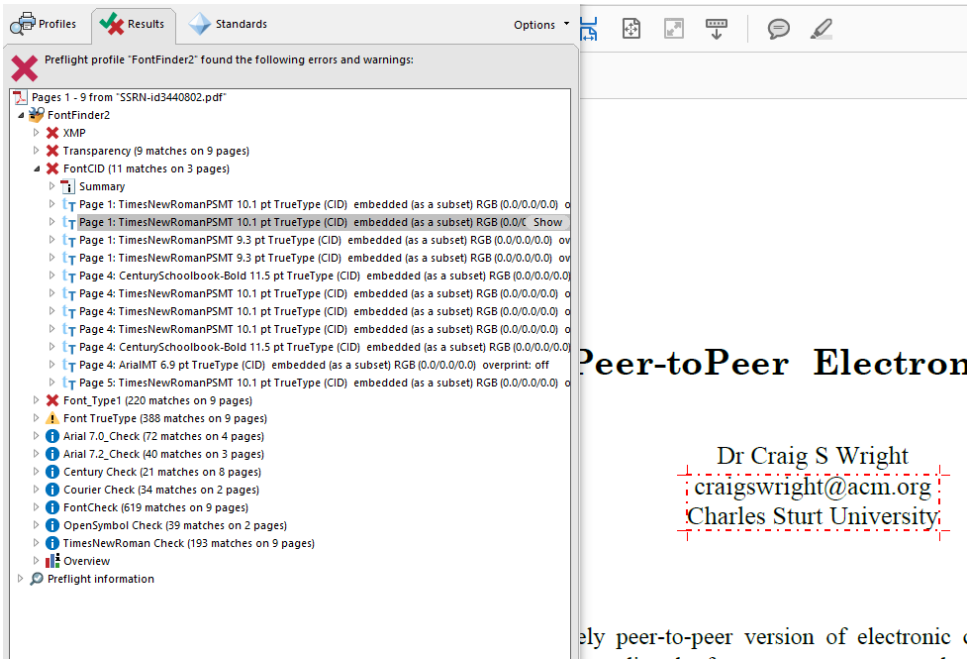




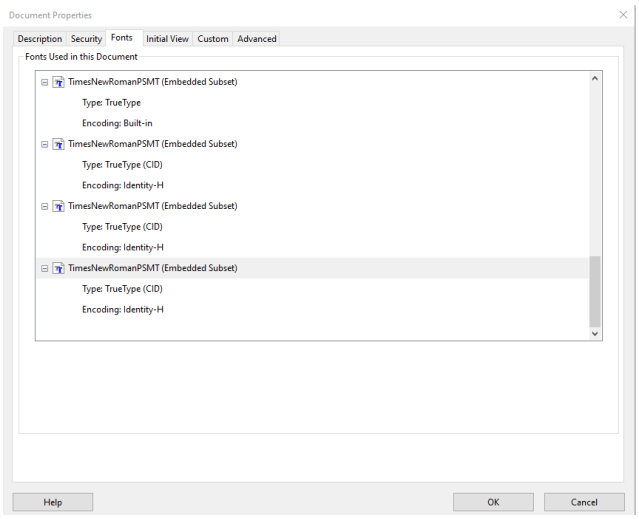
Dette indikerer at deler av teksten i filen hvor det er brukt fontene TimesNewRomanPSMT og ArialMT med TrueType(CID) og Identity-H-enkoding er tekst skrevet i 2017 eller senere.

Innholdsanalyse: Analyserer man hvor i «SSRN-id3440802.pdf» det er brukt fonter med denne enkodingen sammenliknet med «bitcoin.pdf», observerer vi at fontene med forskjellig type og enkoding, nemlig Type: TrueType (CID) med Encoding: Identity-H, er brukt på de steder i filen som har forskjellig tekst sammenliknet med «bitcoin.pdf». Skjermbildet under viser hvordan verktøyet Adobe Preflight fremhever tekst med fonten TimesNewRomanPSMT med *Type: TrueType (CID)* og *Encoding: Identity-H*.

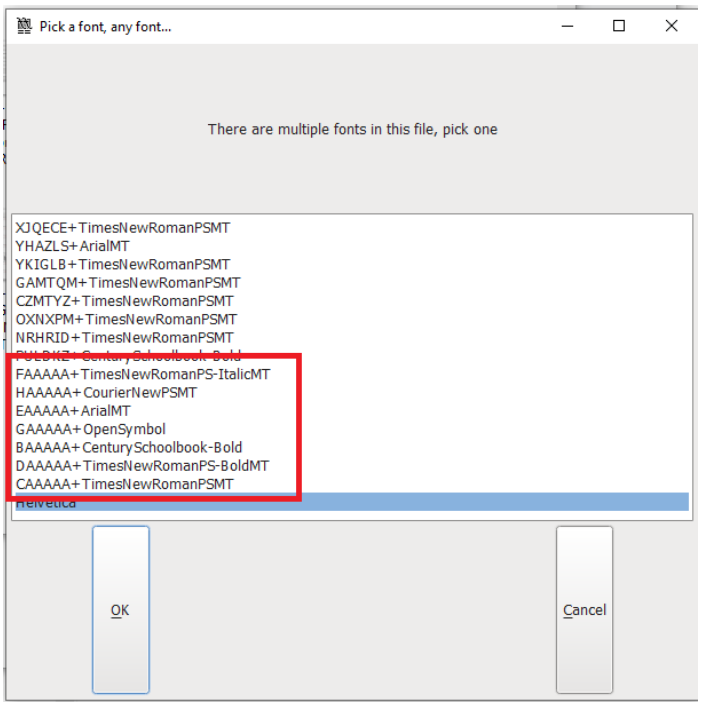
Font-type og enkoding fra «SSRN-id3440802.pdf»



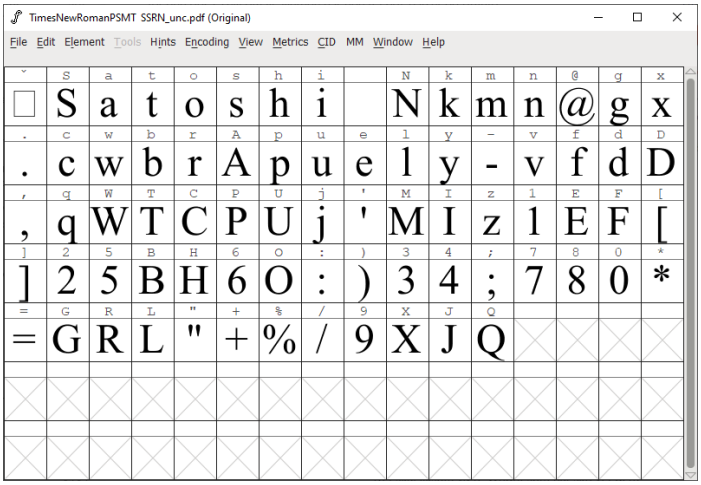
Inspiserer man deretter filens innebygde fonter videre, ser man at filen har tre innebygde subset av TimesNewRomanPSMT med type *TrueType (CID)* og *Identity-H*-enkoding, i tillegg til enkodingen med type *TrueType* og *Built-in* enkoding som man også finner i «bitcoin.pdf».



Fra fontoversikten i FontForge ser man dessuten at «SSRN-id3440802.pdf» har alle de samme innebygde fontene som man observerer i «bitcoin.pdf». Disse er merket i den røde boksen i skjermbildet under:



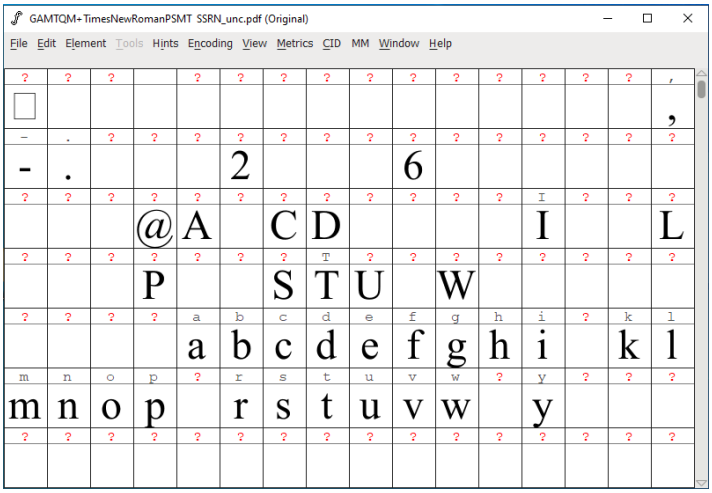
Inspiserer man den siste fonten i den røde boksen i skjermbildet over, *CAAAAA+TimesNewRomanPSMT*, i FontForge, finner man akkurat som «bitcoin.pdf», følgende tegnoversikt:

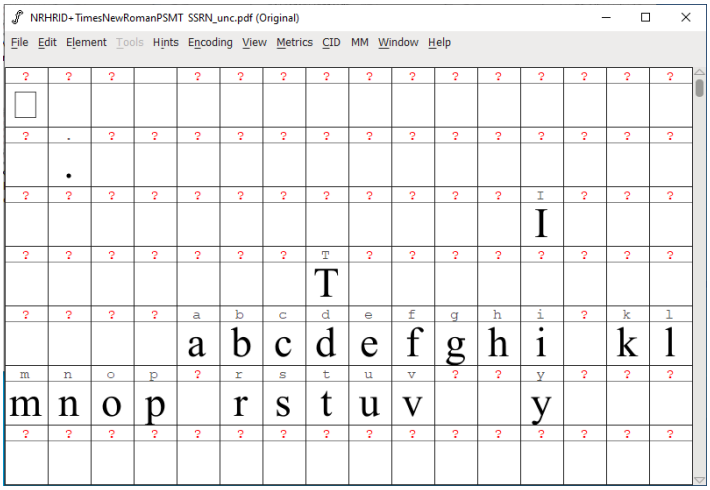


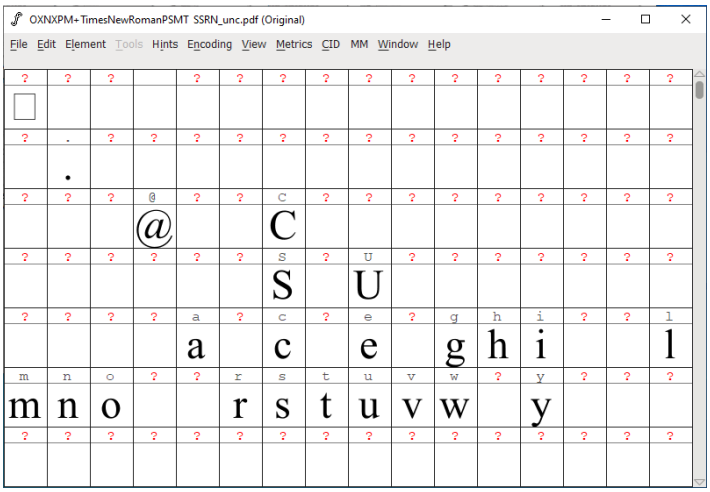
Da alle endringer i fonten TimesNewRomanPSMT i filen «SSRN-id3440802.pdf» er gjort med en annen enkoding har ikke innbyggingen av denne font-enkodingen endret seg, slik at tegnoversikten er helt lik.

Dette er en indikasjon på at «SSRN-id3440802.pdf» er redigert med utgangspunkt i «bitcoin.pdf» eller en tilsvarende fil.

Ser man på tegnoversiktene til de tre innebygde fontene TimesNewRomanPSMT med TrueType (CID) og Identity-H enkoding, observerer man følgende tegnoversikter:

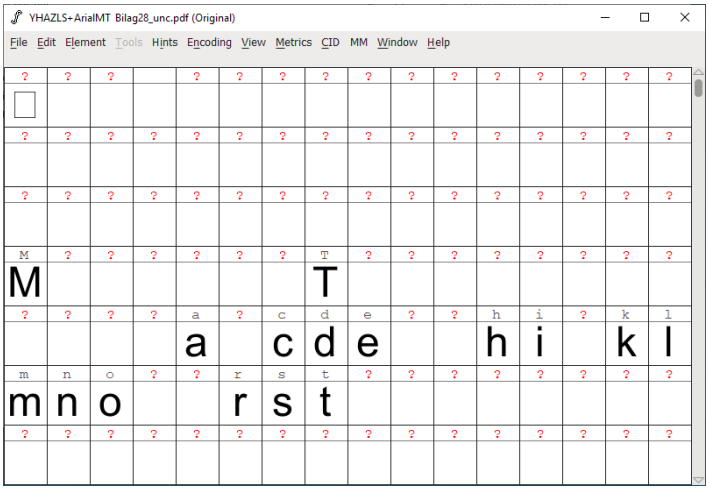




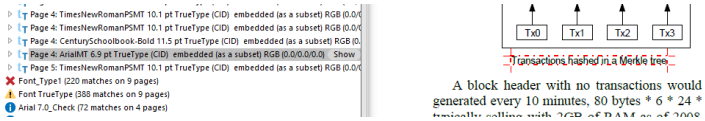


Disse tegnoversiktene representerer all tekst i filen «SSRN-id3440802.pdf» med TimesNewRomanPSMT som er forskjellig fra «bitcoin.pdf».

Det samme gjelder for tegnoversikten man finner for fonten YHAZLS+ Arial MT, som er en font med *TrueType (CID)* og *Identity-H* enkoding:

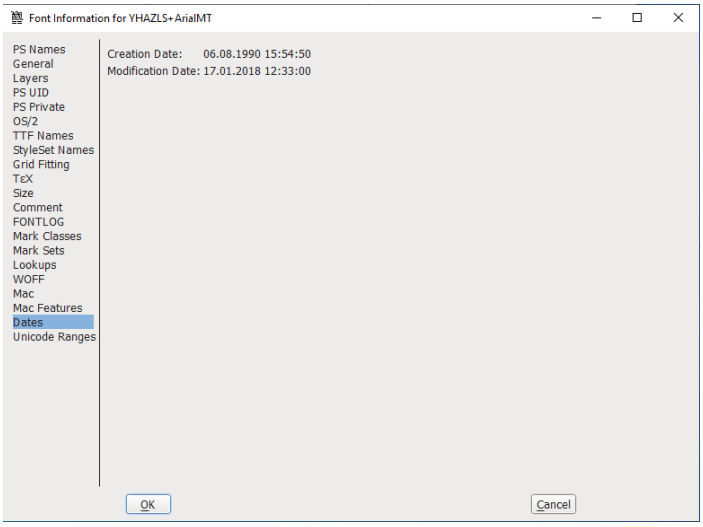


Inspiserer man teksten i filen med denne fonten, ser man at teksten som er forskjellig fra «bitcoin.pdf» leser «*Transactions hashed in a Merkle tree*» på side 4. Dette stemmer overens med tegnene man finner i tegnoversikten over.

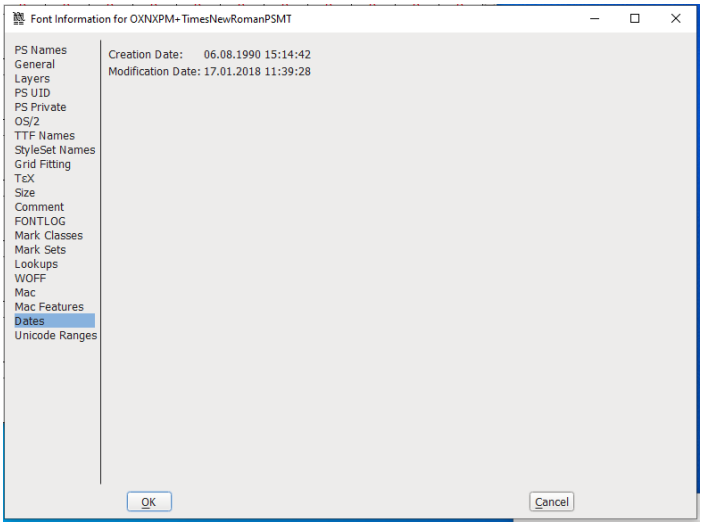


Ser man på font-filenes datoer i FontForge, vil man for fontene med 2017-copyright også observere en endringsdato (**Modification date**) fra 17.januar 2018:

Endringsdato YHAZLS+ ArialMT:



Endringsdato OXNXPM+ TimesNewRomanPSMT:

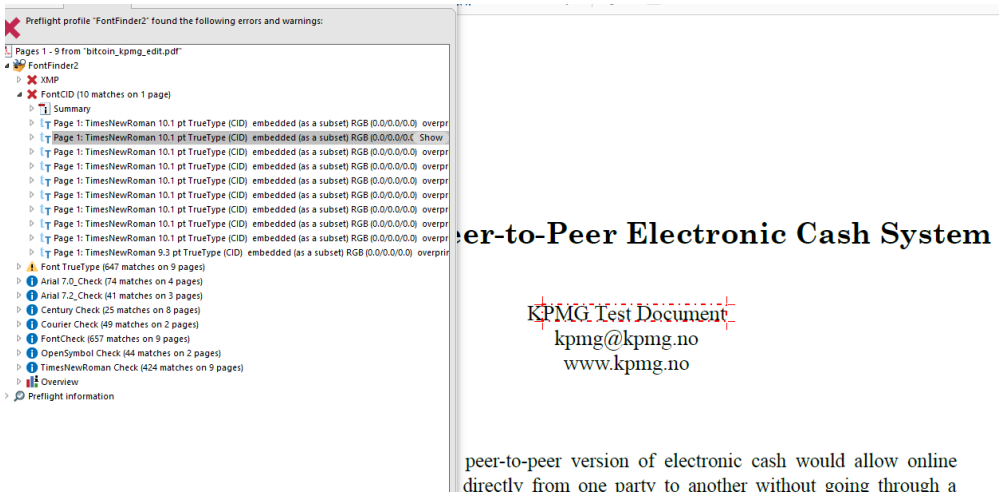


KPMG kan ikke med sikkerhet si hva denne endringsdatoen kommer fra, annet enn at det er en dato fontfilen har registrert for en endring i filens oppbygning.

For alle ulikheter i tekst mellom filene «SSRN-id3440802.pdf» og «bitcoin.pdf», se *Vedlegg 16: Innholdsanalyse «SSRN-id3440802.pdf»*.

KPMG har i tillegg gjennomført en test hvor vi har lastet ned «bitcoin.pdf» fra bitcoin.org og redigert tekst direkte i Adobe Acrobat Pro. Vi observerer at den redigerte teksten får Type: TrueType (CID) og Encoding: Identity-H, i tråd med hva vi observerer i «SSRN-id3440802.pdf». Se skjermbilde fra filen «bitcoin-kpmg-edit.pdf» under.

Font-type og enkoding fra «bitcoin-kpmg-edit.pdf»:



3.2.3.3 bitcoin-draft.pdf

Beskrivelse: Denne filen er et utkast til det endelige whitepaperet som i dag er publisert på bitcoin.org. Denne filen var i sin tid tilgjengelig på bitcoin.org, og Satoshi Nakamoto refererte til Denne filen i en e-post til en kryptografi-e-postliste den 1.november 2018 [27].

Papirstørrelse: Denne filen bruker «Standard Letter»-størrelse (8,5 x 11 tommer), i likhet med «bitcoin.pdf».

Fonter: Denne filen bruker også samme fonter, med lik type og enkoding som «bitcoin.pdf». For alle fonter brukt i denne filen se *Vedlegg 27: Font-oversikt fra tekst: Referansefiler.*

Innholdsanalyse: «bitcoin-draft.pdf» inneholder noen forskjeller i teksten sammenliknet med «bitcoin.pdf». Disse forskjellene er vedlagt i *Vedlegg 17: Innholdsanalyse «bitcoin-draft.pdf»*.

[27] Hentet fra http://satoshinakamoto.me/2008/11/01/bitcoin-p2p-e-cash-paper/ 03.12.21.

3.2.4 Innholds- og formateringsanalyse av bilag

I denne seksjonen vil vi presentere observasjoner fra analyser av fonter, formattering og innhold for enkelte av bilagene der vi har funnet det hensiktsmessig å gjennomføre disse analysene.

3.2.4.1 Bilag 19

Beskrivelse: «Bilag19.pdf» er fremlagt som en pdf-utskrift av en e-post.

Innholdsanalyse: Fra teksten ser det ut til at e-posten er videresendt fra e-postadressen *craig.wright@information-defense.com* den 12. mars 2008 kl. 6:37 PM, da subject-linjen endrer seg fra å inneholde «RE» til «FW». Hvilken e-postadresse e-posten er videresendt til er ikke mulig å fastslå fra denne filen.

Skjermbilde med markering av endring fra «RE:» til «FW:»:

From: Craig S Wright
To: "Ira K"
Subject: FW: Defamation and the diffculties of law on the Internet.
Date: 06 March 2014 22:51:08

-----Original Message-----
From: Craig S Wright [mailto:craig.wright@information-defense.com]
Sent: Wednesday, 12 March 2008 6:37 PM
To: dave kleiman
Subject: FW: Defamation and the diffculties of law on the Internet.

I need your help editing a paper I am going to relase later this year. I have been working on a new form of electronic money. Bit cash, Bitcoin...

You are always there for me Dave. I want you to be a part of it all.

I cannot release it as me. GMX, vistomail and Tor. I need your help and I need a version of me to make this work that is better than me.

Craig

-----Original Message-----
From: dave kleiman [mailto:dave@davekleiman.com]
Sent: Wednesday, 12 March 2008 6:25 PM
To: security-basics@securityfocus.com
Subject: RE: Defamation and the diffculties of law on the Internet.

Hats off to you Craig,

3.2.4.2 Bilag 20

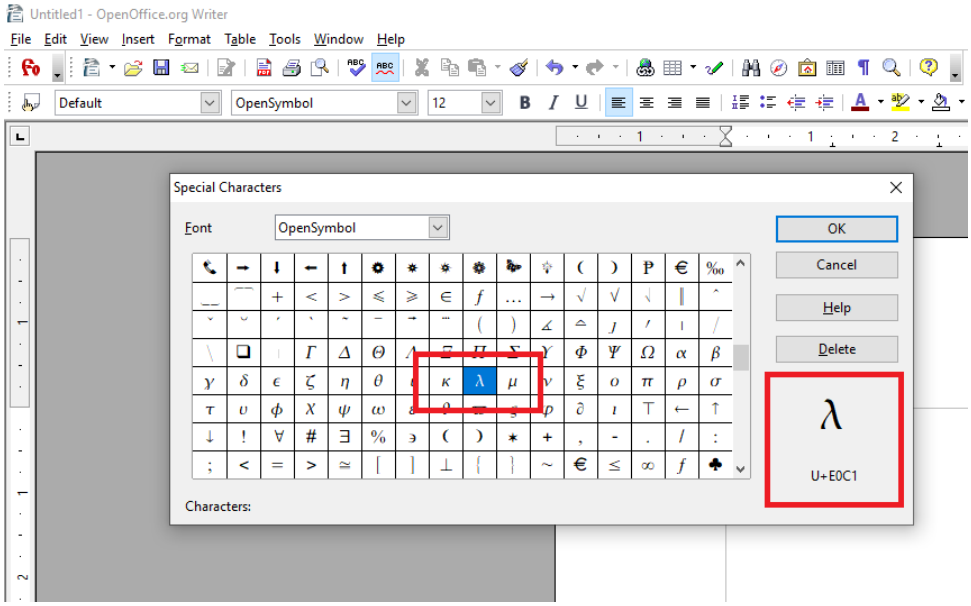
Beskrivelse: «Bilag 20.pdf» er en scan av en versjon av et Bitcoin-whitepaper.

Papirstørrelse: Vi observerer at «Bilag 20.pdf» er en fil i A4-størrelse (8.27 x 11,69 tommer). Hvorvidt originaldokumentet som ble scannet også var A4 eller et annet format, er vanskelig å fastslå, men filen fremlagt er scannet i A4-format. Til sammenlikning er «bitcoin.pdf» i Standard Letter-format.

Fonter: Da «Bilag 20.pdf» er en scannet fil kan det være problematisk å bruke en automatisk font-analyse av innholdet i filen.

I «Bilag 20.pdf» ser vi imidlertid at det på side 7 mangler et symbol for den greske bokstaven «lambda» sammenliknet med «bitcoin.pdf». I «bitcoin.pdf» er denne bokstaven innebygget *(embedded)* fra fonten OpenSymbol. Fra metadataene til «bitcoin.pdf»-filen vet vi at denne ble laget fra Open Office 2.4. Ser man på Unicode-hexadecimal-koden for «lambda» i OpenSymbol i Open Office 2.4 er denne U+E0C1.

Lambda i OpenSymbol i Open Office 2.4:



Symbolet i «Bilag 20.pdf» som mangler kan sees fra skjermbildet under:

Manglende Lambda-tegn:

The recipient waits until the transaction has been added to a block and z blocks have been linked after it. He doesn't know the exact amount of progress the attacker has made, but assuming the honest blocks took the average expected time per block, the attacker's potential progress will be a Poisson distribution with expected value:

$$\square = z\frac{q}{p}$$

To get the probability the attacker could still catch up now, we multiply the Poisson density for each amount of progress he could have made by the probability he could catch up from that point:

$$\sum_{k=0}^{\infty} \frac{\square^{k} e^{-\square}}{k!} \cdot \begin{cases} (q/p)^{(z-k)} & \text{if } k \le z \\ 1 & \text{if } k > z \end{cases}$$

Rearranging to avoid summing the infinite tail of the distribution...

$$1 - \sum_{k=0} \frac{z\square^{k} e^{-\square}}{k!} \quad (1-(q/p)^{(z-k)})$$

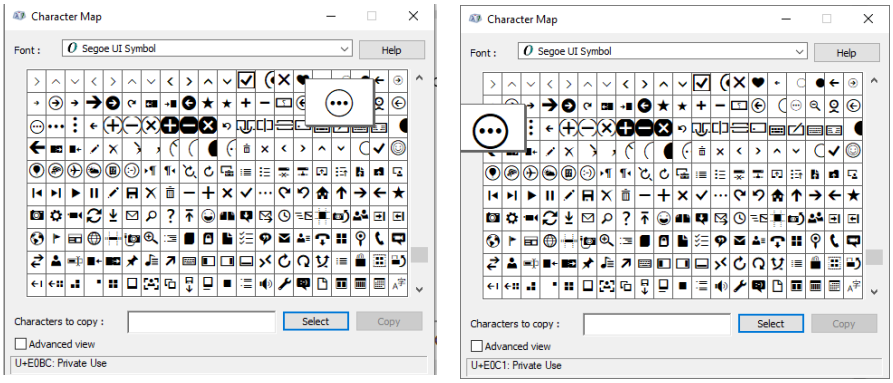
Converting to C code...

```
#include <math.h>
double AttackerSuccessProbability(double q, int z)
{
    double p = 1.0 - q;
    double lambda = z * (q / p);
    double sum = 1.0;
    int i, k;
    for (k = 0; k <= z; k++)
    {
        double poisson = exp(-lambda);
        for (i = 1; i <= k; i++)
            poisson *= lambda / i;
        sum -= poisson * (1 - pow(q / p, z - k));
    }
    return sum;
}
```

7

Dette symbolet tilhører fonten Segoe UI Symbol. Segoe UI Symbol har to symboler som likner på symbolet i «Bilag 20.pdf», som vist av skjermbildene under:

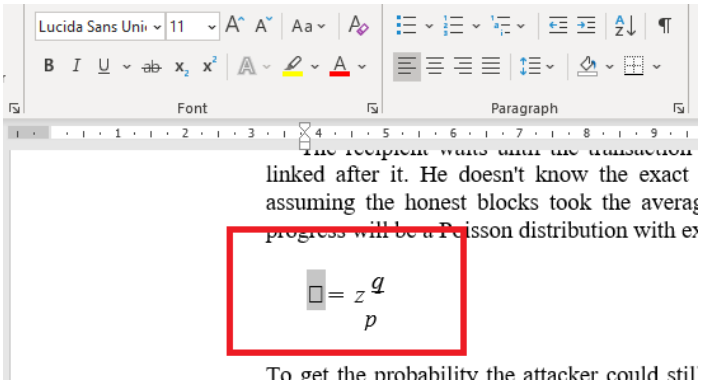
Symboler i Segoe UI Symbol:



Ser man på Unicode-hexadecimal-koden til disse to symbolene er det henholdsvis U+E0BC og U+E0C1. Sistnevnte symbol har den samme Unicode-hexadecimal-koden som «lambda»-symbolet i OpenSymbol.

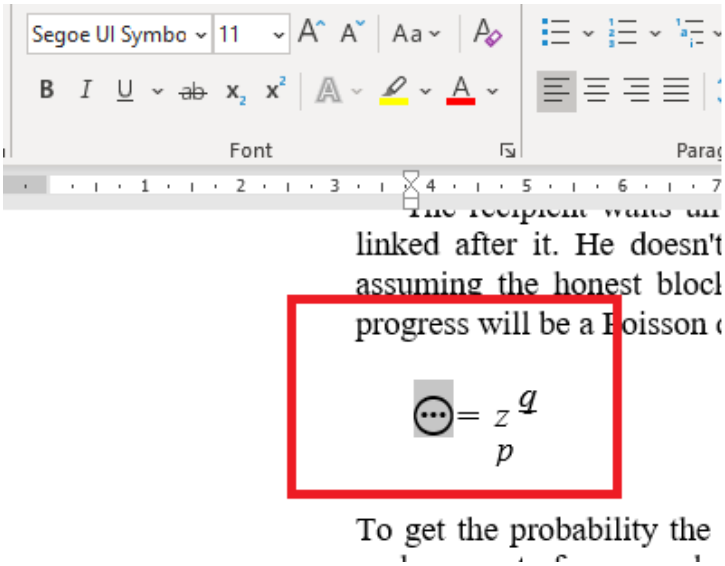
Ved konvertering av eksempelvis filer fra PDF til en teksteditor som Microsoft Word, så vil programvaren bruke fonter installert på enheten til å dekode teksten som kommer fra PDF-filen for å gjøre dokumentet lesbart i Word. Er det fonter fra PDF-filen som ikke er installert på enheten, så vil Word automatisk erstatte denne fonten med en installert font. Når teksten er konvertert kan man selv overstyre hvilken font Word-dokumentet skal erstatte [28]. Om man konverterer en fil med OpenSymbol er ikke dette en font som er forhåndsinstallert med Windows-operativsystemer, og MS Word vil bruke en annen forhåndsinstallert font som erstatning, eksempelvis en av Windows' symbol-fonter Segoe UI Symbol eller Symbol.

KPMG har lastet ned «bitcoin.pdf» og konvertert filen til Word ved hjelp av Adobe Acrobat Pro på en maskin uten fonten OpenSymbol installert (referansefil: «bitcoin-word-export.docx»). I vårt tilfelle prøver Word først å gjengi «lambda»-symbolet i fonten Lucida Sans Unicode:



Endrer man fonten til symbolet til Segoe UI Symbol, får man dette resultatet, i likhet med hva man ser i «Bilag 20.pdf»:

[28] Hentet fra https://wordribbon.tips.net/T012657_Finding_Words_Font_Substitutes.html, 10.12.21.



Segoe UI Symbol er en font som først ble lansert som en del av Windows 7 22.oktober 2009 [29] [30], og som fikk oppdateringer og støtte for ytterligere symboler i senere Windows-versjoner [31] [32]. Dette er etter «bitcoin.pdf» og «bitcoin-draft.pdf» ble gjort offentlig tilgjengelig.

Det fysiske dokumentet som er skrevet ut og deretter scannet som «Bilag 20.pdf» må derfor være fra etter 22.oktober 2009.

Innholdsanalyse:

Da «Bilag 20.pdf» er en scan med begrenset kvalitet er det utfordrende å gjennomføre en automatisk innholdsanalyse mellom dokumentene, selv med OCR-behandling av teksten i «Bilag 20.pdf». Analysen under er derfor gjort manuelt.

Sammenlikner man teksten i «Bilag 20.pdf» med «bitcoin.pdf» er teksten lik, men det finnes enkelte formatteringsforskjeller mellom dokumentene.

- ✓ Gjennomgående ser det ut til at det er små forskjeller i margene.
- ✓ Gjennomgående bruker «Bilag 20.pdf» en annen font på overskrifter. Dette ser ut til å være i fonten Cambria fremfor Century Schoolbook som er brukt i «bitcoin.pdf».
- ✓ På side 2, 4 og 5 er det figurer som har mangler i «Bilag 20.pdf» sammenliknet med «bitcoin.pdf»

[29] Hentet fra https://docs.microsoft.com/en-us/typography/fonts/windows_7_font_list, 10.12.21.

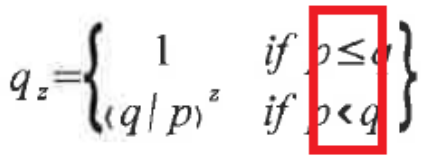
[30] Hentet fra http://www.blacksunsoftware.com/fonts-supplied-with-windows-vista.html, 10.12.21

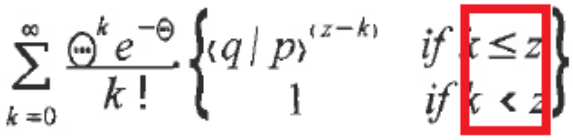
[31] Hentet fra https://docs.microsoft.com/en-us/typography/font-list/segoe-ui-symbol, 10.12.21.

[32] Henter fra https://support.microsoft.com/nb-no/topic/en-oppdatering-for-segoe-symbolfonten-i-brukergrensesnittet-i-windows-7-og-i-windows-server-2008-r2-er-tilgjengelig-0743a473-3afe-e8b2-7c20-54aa430463d6, 10.12.21.

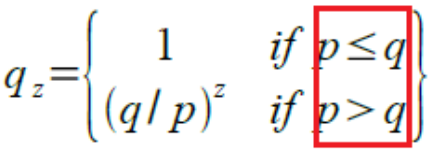
- ✓ På side 6 og 7 er det flere forskjeller i symbol-bruk. I tillegg til forskjellen i bruk av «lambda», som allerede adressert, bruker «Bilag 20.pdf» også et «mindre-enn»-symbol i formlene der «bitcoin.pdf» bruker «større-enn» symbol.

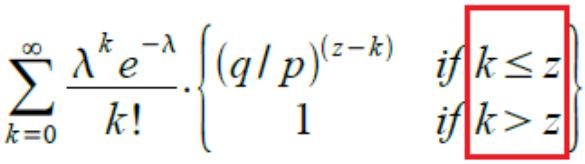
Forskjellig symbol fra side 6 og 7 i «Bilag 20.pdf»:





Forskjellig symbol fra side 6 og 7 i «bitcoin.pdf»:





Når KPMG konverterte «bitcoin.pdf» til Word i filen «bitcoin-export-word.docx» observerer vi at de to stedene der «Bilag 20.pdf» bruker forskjellig symbol fra «bitcoin.pdf», klarer ikke Word tyde disse unicode-tegnene i Lucida Sans Unicode fra OpenSymbol-fonten som «bitcoin.pdf» bruker:

Manglende symbol i «bitocin-export-word.docx»:

$$q_z = \begin{cases} 1 & if\ p \le q \\ \square q / p \square^z & if\ p \square q \end{cases}$$

$$\sum_{k=0}^{\infty} \frac{\square^k e^{-\square}}{k!} \begin{cases} \square q / p \square^{\square z-k \square} & if\ k \le z \\ 1 & if\ k \square z \end{cases}$$

Endrer man disse symbolene til fonten Segoe UI Symbol derimot vil det se slik ut:

$$q_z = \begin{cases} 1 & if\ p \le q \\ \langle q / p \rangle^z & if\ p < q \end{cases}$$

$$\sum_{k=0}^{\infty} \frac{\textcircled{\scriptsize \dots}^k e^{-\textcircled{\scriptsize \dots}}}{k!} \begin{cases} \langle q / p \rangle^{(z-k)} & if\ k \le z \\ 1 & if\ k < z \end{cases}$$

Her observerer vi et «mindre-enn»-tegn, akkurat som i «Bilag 20.pdf», der «bitcoin.pdf» har et «større-enn»-tegn.

Dette kan tyde på at det fysiske dokumentet som «Bilag 20.pdf» er en scan av, ble eksportert fra «bitcoin.pdf» til Word før det ble skrevet ut og scannet til PDF.

For alle forskjeller mellom dokumentene, se *Vedlegg 18: Innholdsanalyse «Bilag 20.pdf».*

3.2.4.3 Bilag 21, 22, 23 og 25

Beskrivelse: Disse bilagene er scans av et håndskrevne notater.

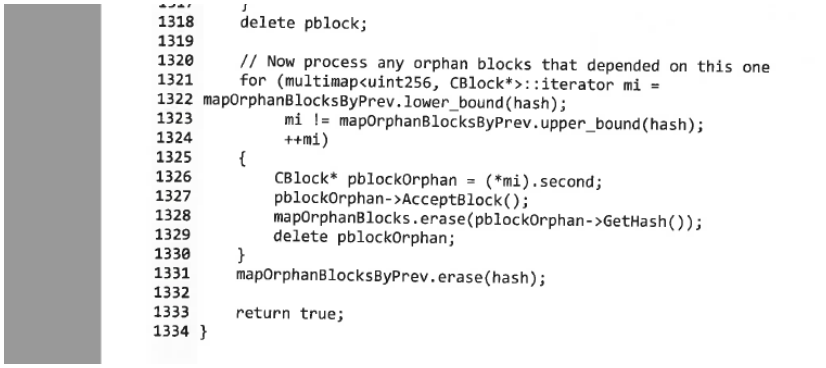
Andre observasjoner: Når innholdet fra de håndskrevne filene er produsert er ikke mulig å fastslå.

3.2.4.4 Bilag 24

Beskrivelse: «Bilag 24.pdf» er fremlagt som en scan av et dokument med kildekode med håndskrevne notater.

Innholdsanalyse: «Bilag 24.pdf» inneholder en kopi av kildekoden til main.cpp, som også finnes offentlig tilgjengelig via et innlegg på bitcointalk.org fra 2013[33]. «Bilag 24.pdf» inneholder kun den første delen av den komplette kildekoden, akkurat som innlegget på forumet, som måtte deles i to separate innlegg på grunn av størrelsesbegrensninger på forumet. Fra «Bilag 24.pdf» ser det ut til at dette er all koden i dokumentet, da det på side 23 står «Page 23 of 23».

Siste linjer med kode på side 23 i «Bilag 24.pdf»



```
1318     delete pblock;
1319
1320     // Now process any orphan blocks that depended on this one
1321     for (multimap<uint256, CBlock*>::iterator mi =
1322 mapOrphanBlocksByPrev.lower_bound(hash);
1323          mi != mapOrphanBlocksByPrev.upper_bound(hash);
1324          ++mi)
1325     {
1326         CBlock* pblockOrphan = (*mi).second;
1327         pblockOrphan->AcceptBlock();
1328         mapOrphanBlocks.erase(pblockOrphan->GetHash());
1329         delete pblockOrphan;
1330     }
1331     mapOrphanBlocksByPrev.erase(hash);
1332
1333     return true;
1334 }
```

Siste linjer med kode fra den første av to forumposter[34] med komplett kode:

Cryddit
Legendary

Activity: 924
Merit: 1071

Re: Bitcoin source from November 2008.
December 23, 2013, 07:32:23 PM

This is the first half of main.cpp. I'm going to post it in two halves, because the forum software allows a

Code:

```
    delete pblock;

    // Now process any orphan blocks that depended on this one
    for (multimap<uint256, CBlock*>::iterator mi = mapOrphanBlocksByPrev.lower_bound(hash);
         mi != mapOrphanBlocksByPrev.upper_bound(hash);
         ++mi)
    {
        CBlock* pblockOrphan = (*mi).second;
        pblockOrphan->AcceptBlock();
        mapOrphanBlocks.erase(pblockOrphan->GetHash());
        delete pblockOrphan;
    }
    mapOrphanBlocksByPrev.erase(hash);

    return true;
}
```

3.2.4.5 Bilag 26

Beskrivelse: «Bilag 26.pdf» er fremlagt som en scan av en artikkel fra JSTOR.org, som det er gjort håndskrevne notater i.

[33] Hentet fra https://bitcointalk.org/index.php?topic=382374.msg4108739#msg4108739, 29.11.2021.

[34] Hentet fra https://bitcointalk.org/index.php?topic=382374.msg4108739#msg4108739, 29.11.2021.

Fonter: Da «Bilag 26.pdf» en scannet fil er det ikke mulig å gjøre en digital analyse av fontene i Adobe Preflight eller FontForge.

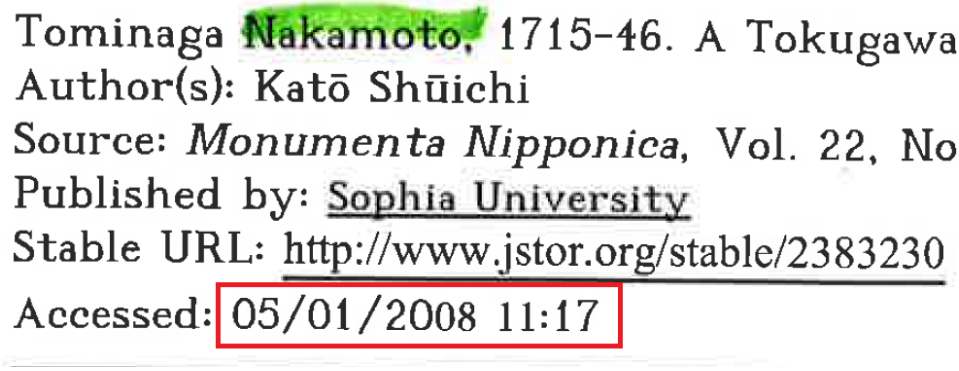
Innholdsanalyse: Filen det er gjort håndskrevne notater i er en artikkel tilgjengelig for kjøp på jstor.org for 12 USD [35]. KPMG har lastet ned denne filen og observerer forskjeller fra formatteringen i referansefil «J-stor-original-article.pdf» og «Bilag 26.pdf». Disse forskjellene er vedlagt i *Vedlegg 19: Innholdsanalyse «Bilag 26.pdf».*

Artikkelen er også offentlig tilgjengelig for nedlasting på https://web.archive.org/web/20160509080944/http://www.gwern.net/docs/tominaga/1967-kato.pdf.

KPMG har lastet ned filen og lagt denne til som referansefil under filnavnet «1967-kato.pdf». Denne filen har lik utforming som «Bilag 26.pdf», men viser en aksess-dato fra 05.01.2015 klokken 22:17, til forskjell fra «Bilag 26.pdf» som viser en aksess-dato 05.01.2008 klokken 11:17.

KPMG observerer at begge filene har en aksessdato den 05. januar, men fra forskjellige år. I tillegg til å være aksessert 17 minutter over henholdsvis 22:00 og 11:00.

Ser man nærmere på teksten for aksessdato i «Bilag 26.pdf» kan det se ut til at sifrene «08» i datoen og «11» i klokkeslettet er forskjellig fra de resterende sifrene, som vist i skjermbildet under:


Tominaga Nakamoto, 1715-46. A Tokugawa
Author(s): Katō Shūichi
Source: *Monumenta Nipponica*, Vol. 22, No
Published by: Sophia University
Stable URL: http://www.jstor.org/stable/2383230
Accessed: 05/01/2008 11:17

Ser man spesielt på 1-tallene i klokkeslettet, ser man forskjeller fra utformingen på 1-tallene som angir timer og minutter, som tyder på at dette er skrevet i en annen font:


1/2008 11:17

[35] Lastet ned fra https://www.jstor.org/stable/2383230 23.11.2021

På lik linje observerer KPMG forskjeller i «0»-tallene i årstallet:

Dette kan tyde på at sifrene «08» i årstallet og «11» er redigert. Dette er også de sifrene som er forskjellig fra teksten i «1967-kato.pdf».

Redigerer man teksten i filen «1967-kato.pdf» i et verktøy som Adobe Acrobat Pro, ser man at teksten er skrevet i fonten «Code2000». Forsøker man å redigere med denne fonten i Adobe Acrobat, vil programvaren automatisk redigere teksten i fonten Minion Pro (selv om man har fonten Code2000 installert på maskinen). Dette er da teksten man observerer:

01/2008 11:17 (KPMG-eksempel)

KPMG observerer ulikheter mellom sifrene i denne teksten på lik linje med hva vi observerer i «Bilag 26.pdf».

I bunnteksten observerer KPMG forskjell mellom filene i dateringen og IP-adresse-informasjonen. For forskjellene mellom filene, se *Vedlegg 19: Innholdsanalyse «Bilag 26.pdf»* for disse forskjellene.

3.2.4.6 Bilag 27

Beskrivelse: «Bilag 27.odt» fremstår å være en versjon av et whitepaper.

Papirstørrelse: Denne filen bruker papirstørrelse A4 (210 x 297 mm), som i hovedsak er standardstørrelsen til alle land utenom USA og Canada, jf. ISO 216 [36]. Når man produserer dokumenter til PDF vil papirstørrelsen følge fra papirstørrelsen til filen i applikasjonen man produserer den fra. Dette gjelder også andre veien når man konverterer et dokument fra PDF til eksempelvis Microsoft Word. Det finnes ikke en innebygget konvertering fra eksempelvis Adobe Acrobat til Open Office.

Fonter: I Open Office kan man installere en extension «TestFonts» som viser alle fontene et dokument innehar [37]. Fra *Vedlegg 20: Innholdsanalyse «Bilag 27.odt»* ser vi at «Bilag 27.odt» bruker

[36] Hentet fra https://www.papersizes.org/a-paper-sizes.htm, 02.12.2021.

[37] Hentet fra https://extensions.openoffice.org/fr/project/testfonts, 01.12.2021.

fontene Arial, Calibri, Cambria Math, Century Schoolbook, Courier New og Times New Roman. Til sammenlikning bruker ikke «bitcoin.pdf» fontene Calibri og Cambria Math.

«Bilag 27.odt» bruker til forskjell fra «bitcoin.pdf» fonten Century Schoolbook i brødteksten, der «bitcoin.pdf» bruker TimesNewRoman.

I likhet med «SSRN-id3440802.pdf» inneholder også PDF-versjonen av «Bilag 27.odt» bruk av både font-type TrueType og TrueType(CID), men med henholdsvis Ansi og Identity-H enkoding. KPMG observerer at PDF-versjonen av «Bilag 27.odt», altså filen «2008-05-06_B027_-_TimeCoin_Peer-to-Peer_Electronic_Cash_System__Dr._Craig_S._Wright.pdf.pdf» ikke inneholder fonten Cambria Math som påvist i Open Office. Se *Vedlegg 20: Innholdsanalyse «Bilag 27.odt»* og *Vedlegg 28: Font-oversikt fra tekst: Bilag 27, 28* & 32 for detaljer.

Innholdsanalyse:

For å enklere sammenlikne teksten med «bitcoin.pdf» har KPMG brukt bilaget «2008-05-06_B027_-_TimeCoin_Peer-to-Peer_Electronic_Cash_System__Dr._Craig_S._Wright.pdf.pdf», som er PDF-versjonen av «Bilag 27.odt».

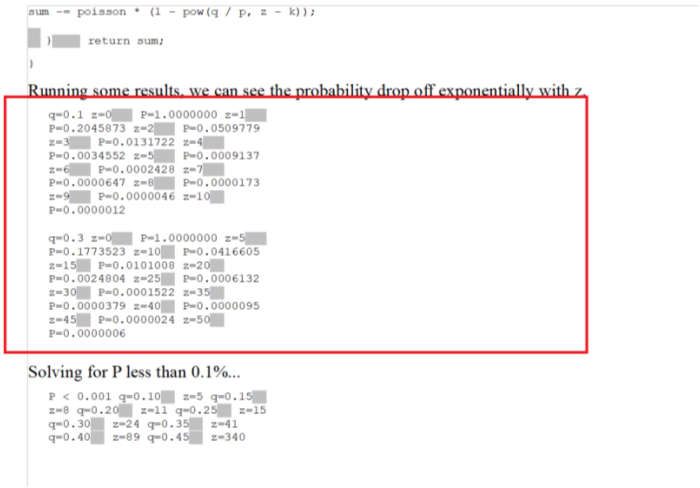
«Bilag 27.odt» inneholder ingen figurer slik «bitcoin.pdf» gjør, men i kapittel 2 «Transactions» er det et lite objekt som Open Office-applikasjonen ikke har klart å tyde. Dette objektet bruker fonten Calibri, som har vært standardfonten til blant annet Microsoft Word siden 2007[38]. Se *Vedlegg 20: Innholdsanalyse «Bilag 27.odt»* for detaljer.

Sammenliknet med «bitcoin.pdf» inneholder også «Bilag 27.odt» et nytt avsnitt i kapittel 6: «Incentive». Dette avsnittet finner man imidlertid igjen i kapittel 4 i «bitcoin.pdf».

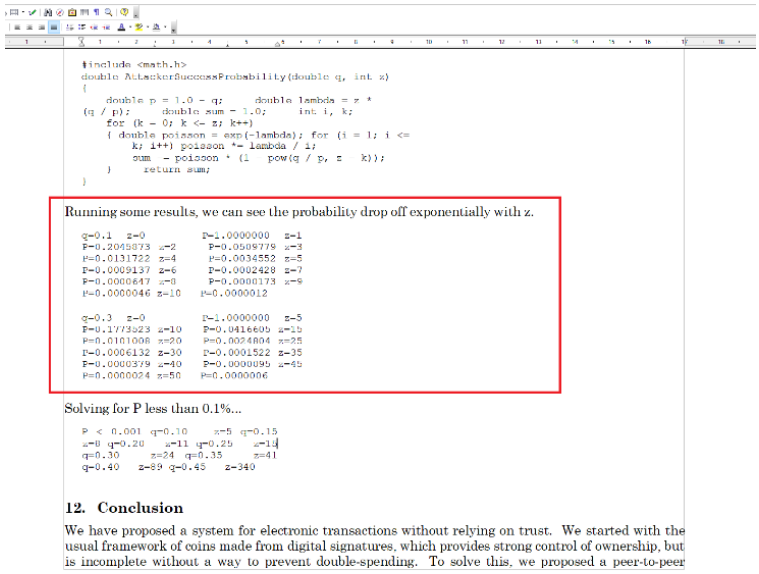
På side 6 har tallverdiene i «Bitcoin 27.odt» en annerledes tabellformatering enn i «bitcoin.pdf». Ser man på skjermbildet under fra filen «PDF til Word til odt (Acrobat Reader).odt» viser dette tabellformattering fra samme seksjon fra «bitcoin.pdf». «PDF til Word til odt (Acrobat Reader).odt» er en fil KPMG har konvertert til Word med Adobe Acrobat Reader, og deretter åpnet direkte i Open Office. KPMG observerer likheter i formatteringen av tabellen med «Bilag 27.odt».

Tabellformattering «PDF til Word til odt (Acrobat Reader).odt» åpnet i Open Office:

[38] Hentet fra https://www.microsoft.com/en-us/microsoft-365/blog/2021/04/28/beyond-calibri-finding-microsofts-next-default-font/, 29.11.2021.



Tabellformattering «Bilag 27.odt»:



For fullstendige forskjeller mellom «Bilag 27.odt» og «bitcoin.pdf» se *Vedlegg 20: Innholdsanalyse «Bilag 27.odt».*

Marger: Dette dokumenter har marger på 0,79 tommer (2 cm) for topp, bunn, venstre og høyre. Dette er i tråd med standard marg-størrelse for Open Office dokumenter[39].

[39] Hentet fra https://smallbusiness.chron.com/set-default-margins-openoffice-63796.html, 30.11.2021.

3.2.4.7 Bilag 28

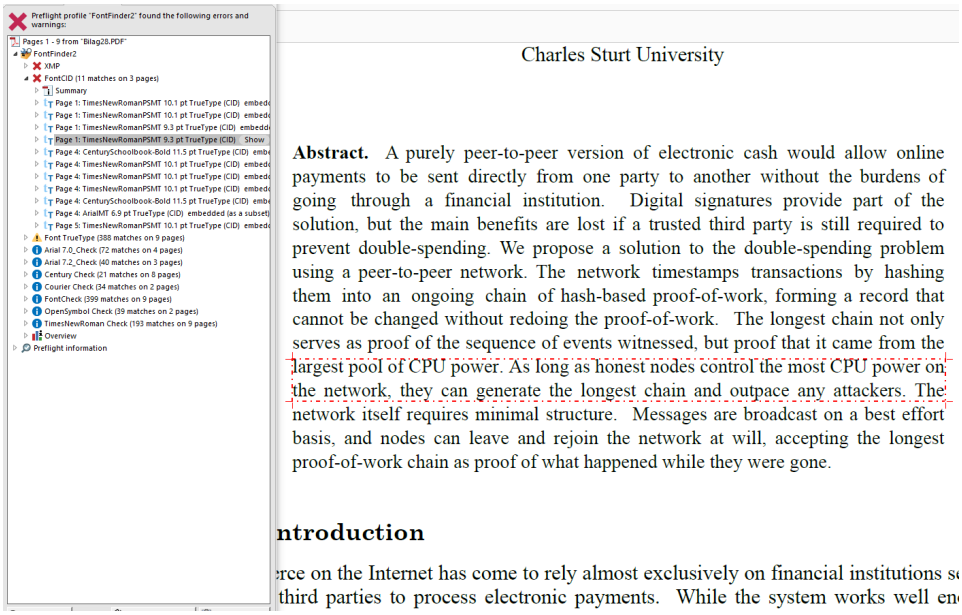
Beskrivelse: «Bilag 28.pdf» fremstår å være en versjon av et whitepaper.

Papirstørrelse: Denne filen har «Standard Letter» papirstørrelse.

Fonter: Filen «Bilag 28.pdf» bruker i likhet med «SSRN-id3440802.pdf» de samme fontene, med den samme typen og enkodingen. Det eneste unntaket er Helvetica-fonten som er en del av bunnteksten i «SSRN-id3440802.pdf». Se *Vedlegg 21: Innholdsanalyse «Bilag 28.pdf»* og *Vedlegg 28: Font-oversikt fra tekst: Bilag 27, 28* & 32 for fullstendig liste over fonter i filen.

Skjermbildet under viser bruk av CID-enkoding. Dette er gjennomgående, i likhet med «SSRN-id3440802.pdf», der teksten er forskjellig fra «bitcoin.pdf»

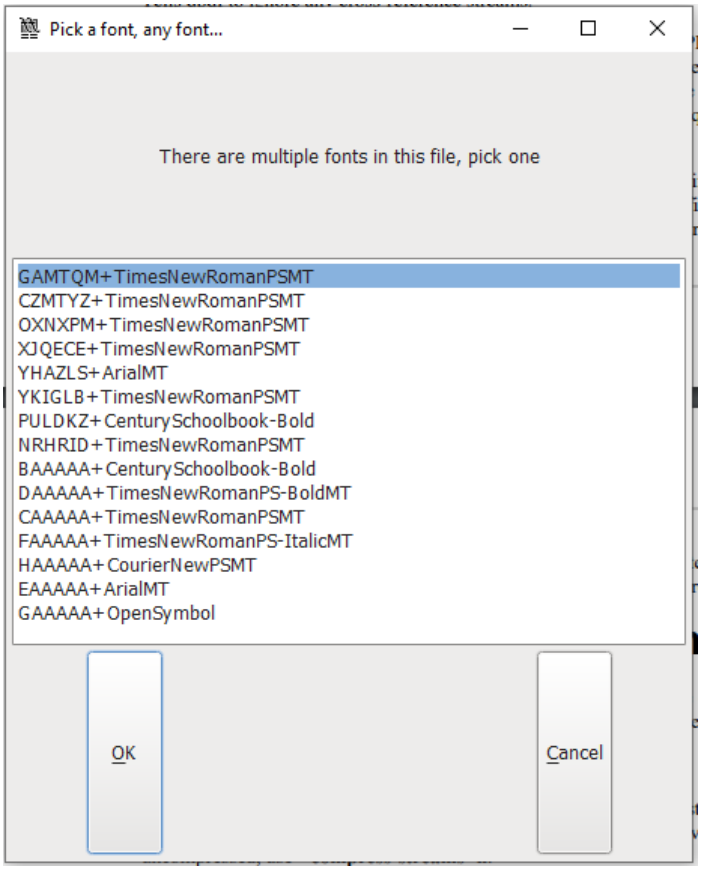
Font-type og enkoding fra «Bilag 28.pdf»:



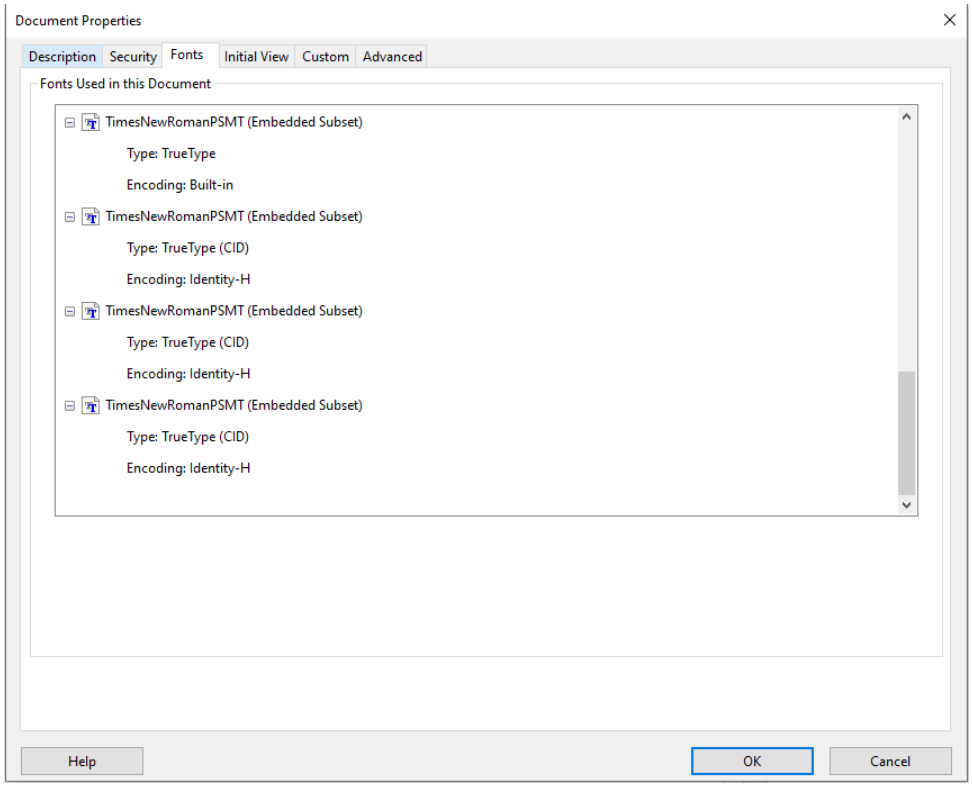
Undersøker man fontene i denne filen med lik fremgangsmåte som «SSRN-id3440802.pdf», med bruk av verktøyet QPDF for dekomprimering av filens font-datastrømmer, observerer man at fontene har den samme copyright-informasjonen. Først dekomprimerer vi «Bilag 28.pdf» til en ny fil «Bilag 28_unc.pdf»:

```
$ ../qpdf-10.4.0/bin/qpdf --stream-data=uncompress Bilag28.pdf Bilag28_unc.pdf
```

Deretter kan vi trekke ut fontene fra den dekomprimerte filen i FontForge:

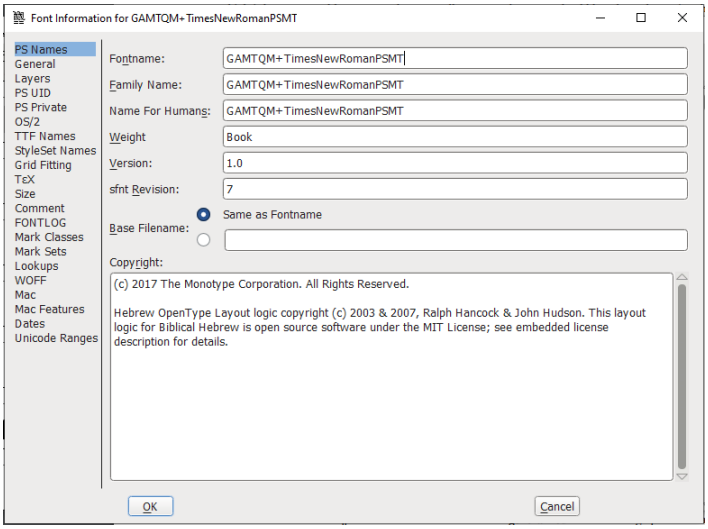


I likhet med «SSRN-id3440802.pdf» observerer vi at «Bilag 28.pdf» har tre innebygde subset av TimesNewRomanPSMT med type *TrueType (CID)* og *Identity-H*-enkoding, i tillegg til enkodingen med type *TrueType* og *Built-in* enkoding som man også finner i «bitcoin.pdf». Skjermbildet under fra Adobe Acrobat viser også dette:

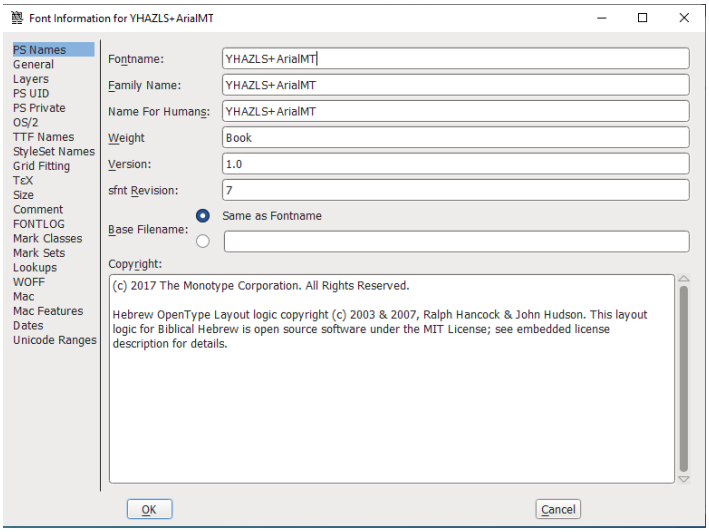


Inspiserer man de innebygde fontene med TrueType(CID) og Identity-H-enkoding, ser man at disse har en copyright fra 2017. Akkurat som observasjonene i «SSRN-id3440802.pdf».

GAMTQM+ TimesNewRomanPSMT med 2017-copyright:

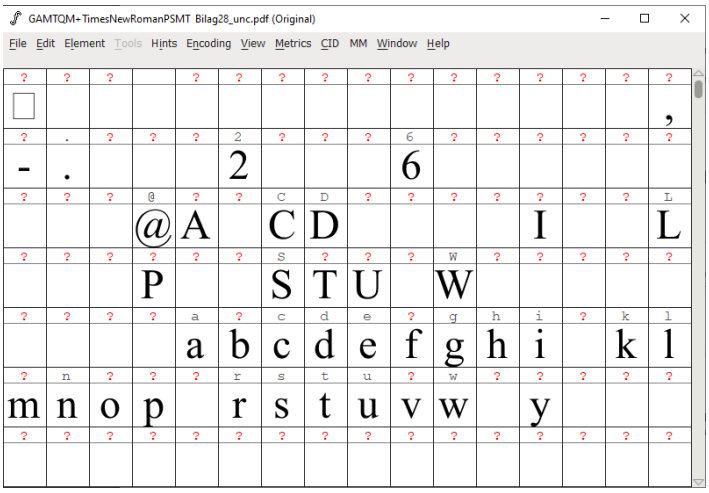


YHAZLS+ ArialMT med 2017-copyright:

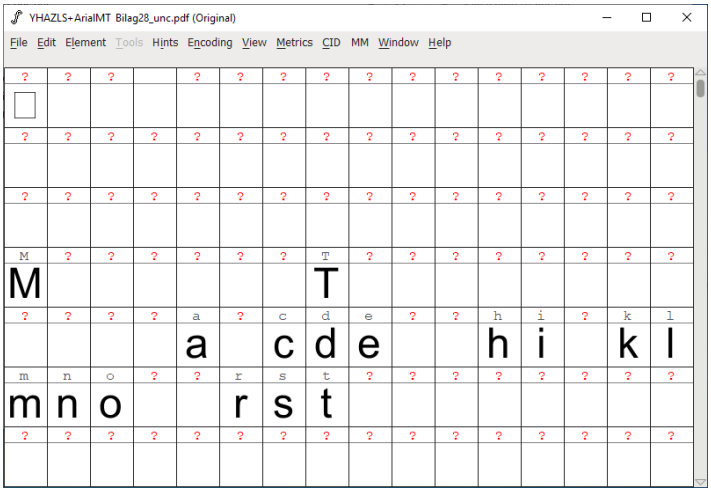


Inspiserer man tegn-oversikten til disse to fontene, observerer vi det samme som for «SSRN- id3440802.pdf»:

Tegnoversikt GAMTQM+ TimesNewRomanPSMT:



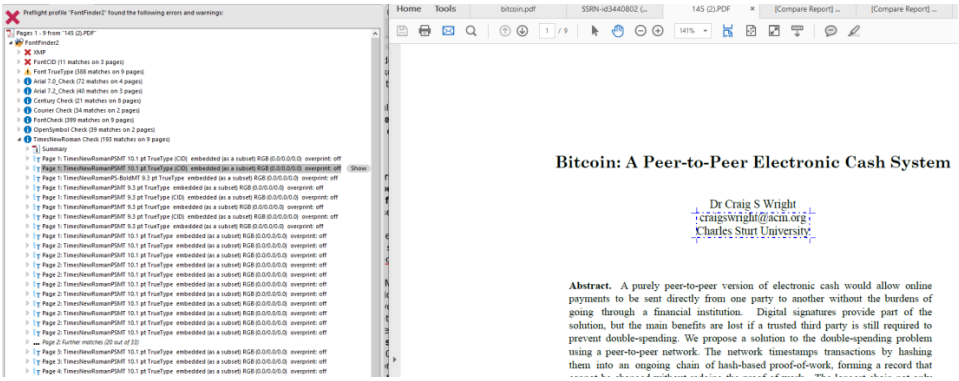
Tegnoversikt YHAZLS+ ArialMT:



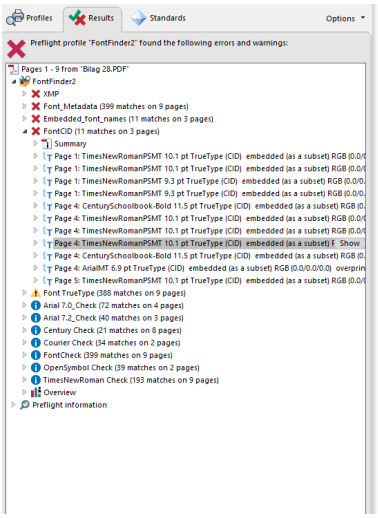
Disse to tegnoversiktene representerer de forskjellene disse to fontene står for sammenlikner man «Bilag 28.pdf» og «bitcoin.pdf».

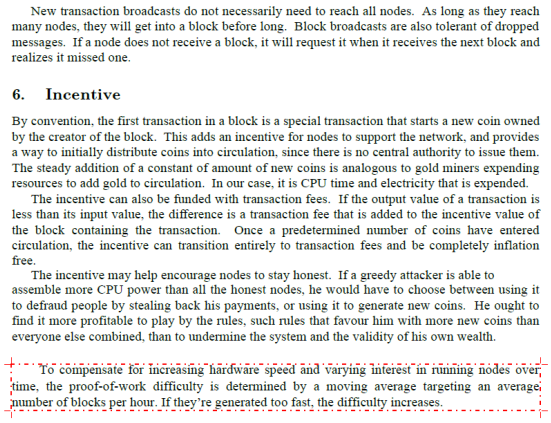
Innholdsanalyse:

Analyserer man hvor det er brukt fonter med denne enkodingen i «Bilag 28.pdf» sammenliknet med «bitcoin.pdf», observerer vi at det er på de steder i filen som har forskjellig tekst sammenliknet med «bitcoin.pdf».



Videre har «Bilag 28.pdf», i likhet med «SSRN-id3440802.pdf» også et ekstra avsnitt i kapittel 6. Dette avsnittet finnes imidlertid også i kapittel 4 i både «Bilag 28.pdf» og «bitcoin.pdf». Avsnittet i kapittel 6 er med en annen enkoding enn avsnittet i kapittel 4.





New transaction broadcasts do not necessarily need to reach all nodes. As long as they reach many nodes, they will get into a block before long. Block broadcasts are also tolerant of dropped messages. If a node does not receive a block, it will request it when it receives the next block and realizes it missed one.

6. Incentive

By convention, the first transaction in a block is a special transaction that starts a new coin owned by the creator of the block. This adds an incentive for nodes to support the network, and provides a way to initially distribute coins into circulation, since there is no central authority to issue them. The steady addition of a constant of amount of new coins is analogous to gold miners expending resources to add gold to circulation. In our case, it is CPU time and electricity that is expended.

The incentive can also be funded with transaction fees. If the output value of a transaction is less than its input value, the difference is a transaction fee that is added to the incentive value of the block containing the transaction. Once a predetermined number of coins have entered circulation, the incentive can transition entirely to transaction fees and be completely inflation free.

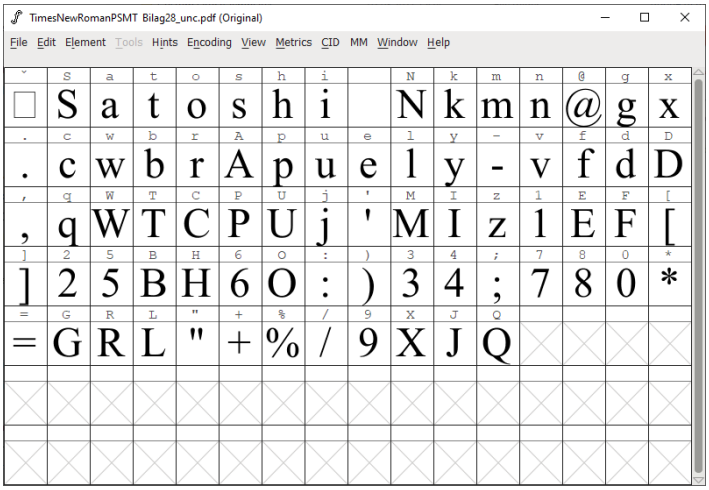
The incentive may help encourage nodes to stay honest. If a greedy attacker is able to assemble more CPU power than all the honest nodes, he would have to choose between using it to defraud people by stealing back his payments, or using it to generate new coins. He ought to find it more profitable to play by the rules, such rules that favour him with more new coins than everyone else combined, than to undermine the system and the validity of his own wealth.

To compensate for increasing hardware speed and varying interest in running nodes over time, the proof-of-work difficulty is determined by a moving average targeting an average number of blocks per hour. If they're generated too fast, the difficulty increases.

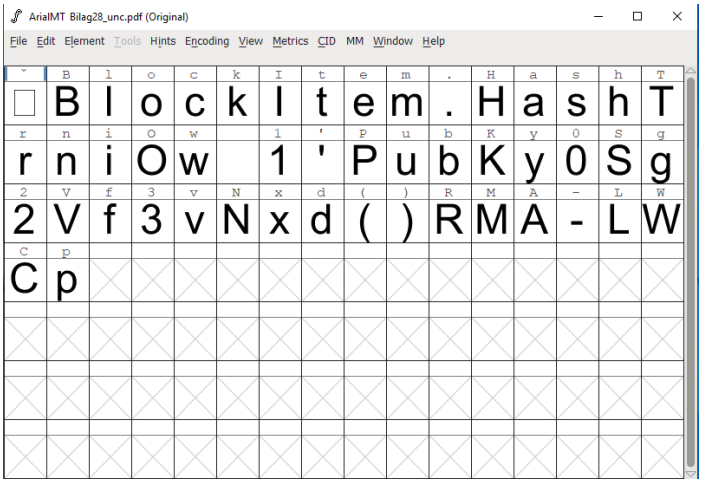
7. Reclaiming Disk Space

«Bilag 28.pdf» inneholder også de samme innebygde fontene, med samme tegnoversikter som vi observerer i «bitcoin.pdf».

Tegnoversikt TimesNewRomanPSMT fra «Bilag 28.pdf»:

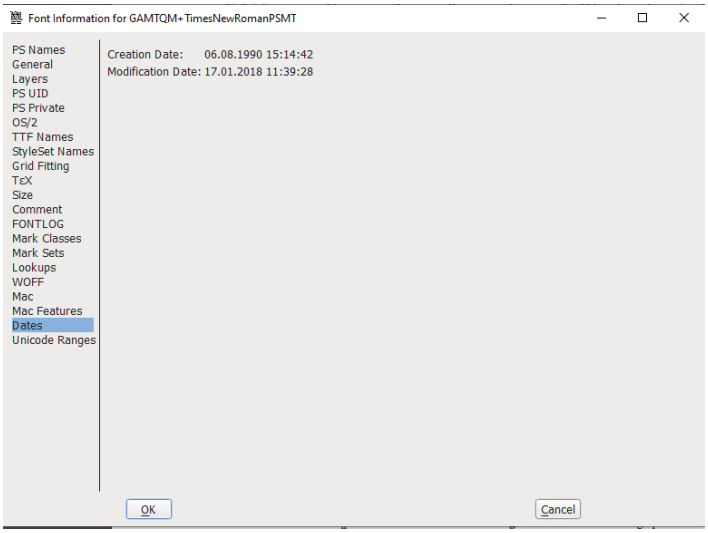


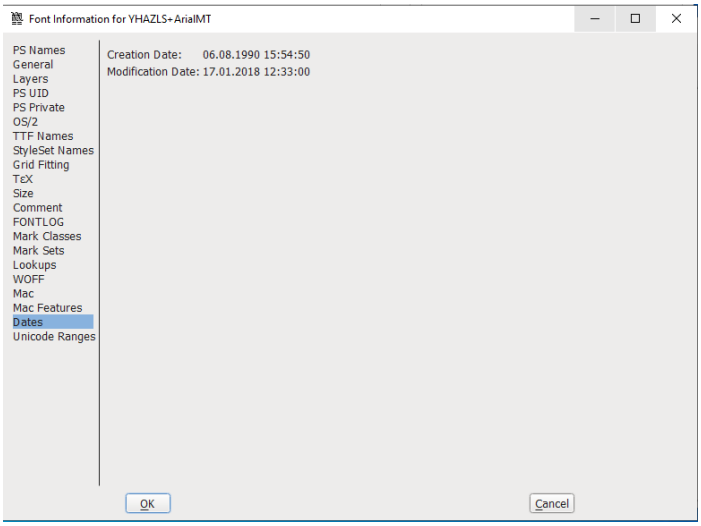
Tegnoversikt ArialMT fra «Bilag 28.pdf»:



Dette er en indikasjon på at «Bilag 28.pdf» er redigert med bakgrunn i «bitcoin.pdf».

Fontene med 2017-copyright har i likhet med «SSRN-id3440802.pdf», også i «Bilag 28.pdf» registrert en endringsdato til font-filene den 17. januar 2018. Hva denne datoen innebærer kan ikke KPMG si med eksakt sikkerhet, annet enn at det er en dato for en endring til font-filen:





Se *Vedlegg 21: Innholdsanalyse «Bilag 28.pdf»* for fullstendige forskjeller mellom de to dokumentene.

3.2.4.8 Bilag 29

Beskrivelse: «Bilag 29.pdf» er et flytskjema.

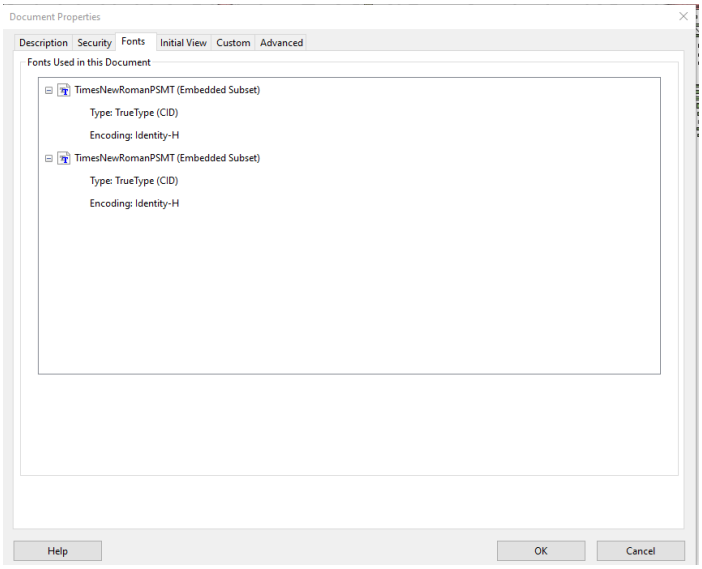
Innholdsanalyse: Teksten i filen er svært liten og vanskelig å tyde. Men ved å zoome tilstrekkelig inn i en programvare som Adobe Acrobat Pro 17 er det mulig å tyde følgende tekst i headeren:

«Copyright © 2009 Satoshi Nakamoto. Distributed under the MIT/X11 software license, see the accompanying file license.exe or http://www.opensource.org/licenses/mit-license.php."

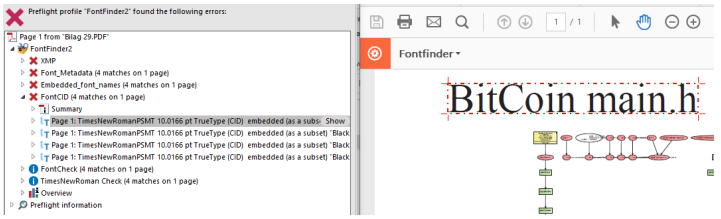
Denne teksten samsvarer ikke med den copyrighten som er fremsatt i de andre kildekode-bilagene fra prosesskrivet til WR, men heller med original-koden som er offentlig tilgjengelig. Se *Vedlegg 22: Innholdsanalyse «Bilag 29.pdf»* for detaljer.

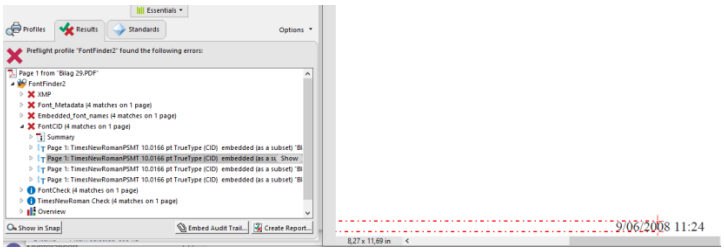
Fonter: Undersøker man enkodingen til fontene i filen er det bygget inn to subset av fonten TimesNewRoman med Type: TrueType(CID) og Encoding: Identity-H.

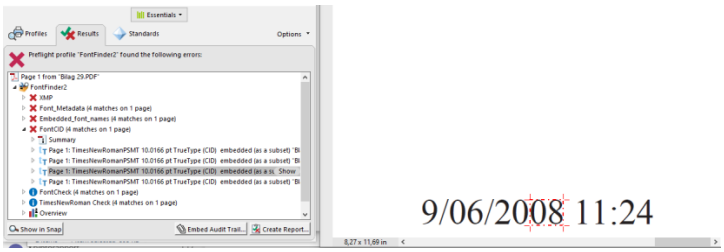
1216 1216



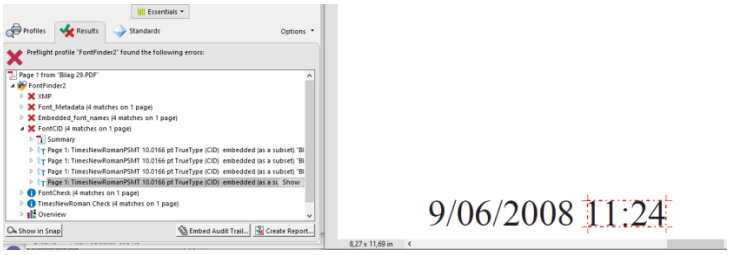
I Adobe Preflight ser man at overskriften og tidsstemplingen nede i høyre hjørne har denne enkodingen:





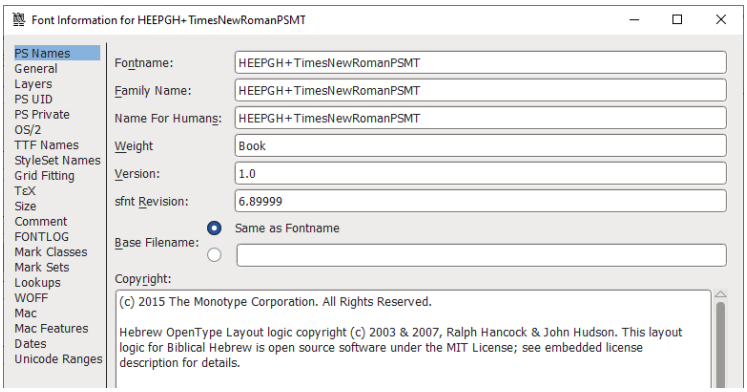


1216

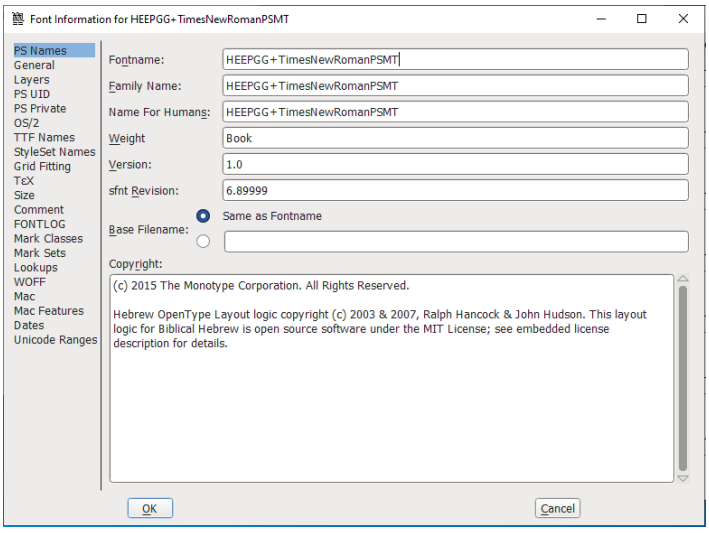


Ser man videre på disse to innebygde fontenes metadata i FontForge, ser man at begge subsettene inneholder en copyright fra 2015.

HEEPGH+ TimesNewRomanPSMT fra «Bilag 29.pdf»:



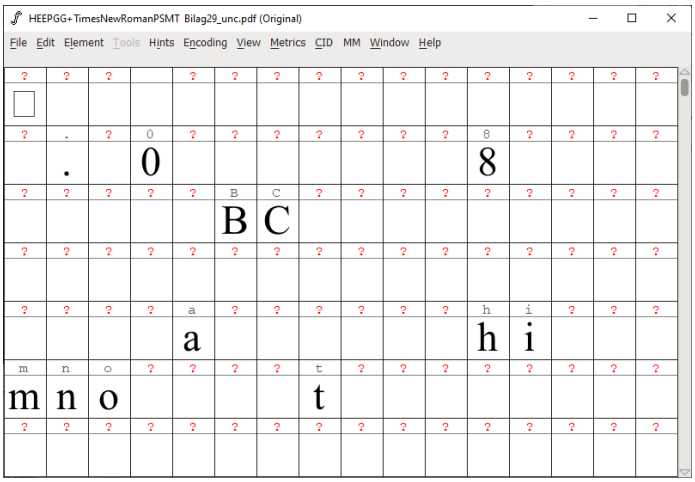
HEEPGG+ TimesNewRomanPSMT fra «Bilag 29.pdf»:

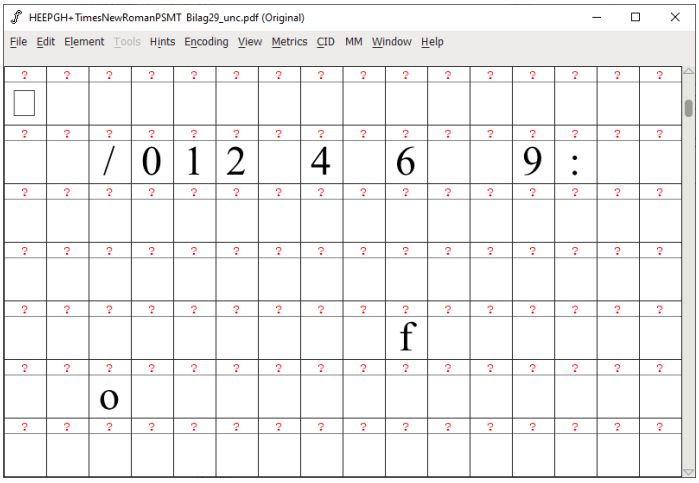


1218

Dette er metadata som er i tråd med filens metadata som sier at filen er laget ved hjelp av Adobe Acrobat Distiller 15, som ble lansert i 7.april 2015 [40].

Ser man på tegn-oversikten til disse to fontene ser man at tegnoversiktene representerer all teksten i «Bilag 29.pdf»:

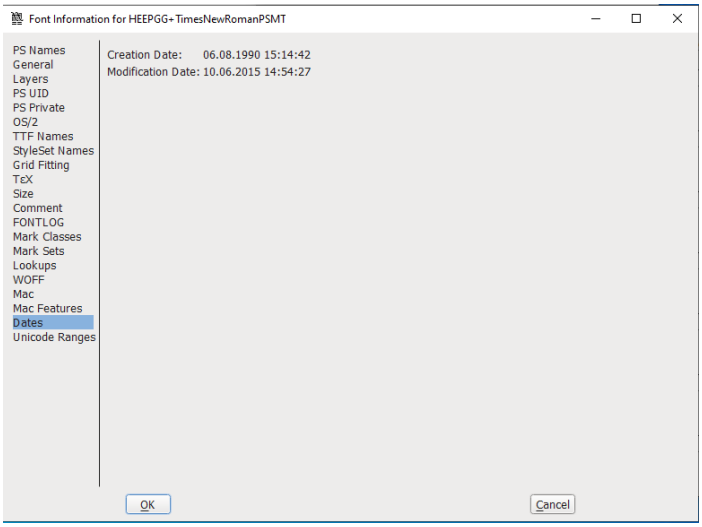


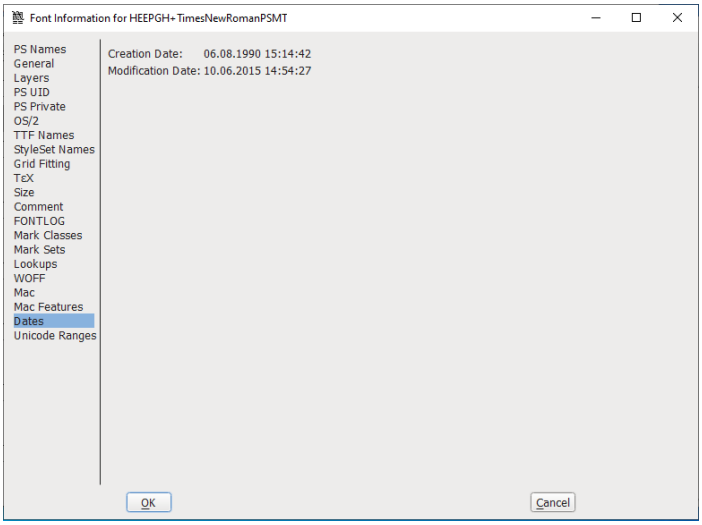


De innebygde fontene i «Bilag 29.pdf» har også registrert en endringsdato til font-filene den 10. juni 2015. Hva denne datoen innebærer kan ikke KPMG si med eksakt sikkerhet, annet enn at det er en dato for en endring til font-filen:

[40] Hentet fra https://web.archive.org/web/20150415142616/https://itunes.apple.com/us/app/adobe-acrobat-dc-pdf-reader/id469337564?mt=8, 07.12.21

1218





3.2.4.9 Bilag 30

Beskrivelse: «Bilag 30.doc» er et tekstdokument med det som ser ut til å være notater.

Papirstørrelse: Filens papirstørrelse er A4, som er standardstørrelse for alle land utenom USA og Canada, jf. ISO-216.

Marger: Filens marger er 1 tomme (2,54 cm) for både topp, bunn, venstre og høyre. Standard margstørrelse for MS Office Word 97-2003 dokumenter er 1-tommes (2,54 cm) marger for topp og bunn, og 1,25 tommes (3,17cm) marger for venstre og høyre. Standardstørrelsen for marger for MS Word fra 2007 og senere er 1-tommes marger for topp, bunn, venstre og høyre.

Fonter: Un*dersøker man enkodingen til fontene i PDF-versjonen av filen er det brukt Arial og Calibri med Type: TrueType og Encoding: Ansi, i tillegg til at det er brukt fonten Symbol med Type: TrueType(CID) og Encoding: Identity-H enkoding. Beror man seg på samme argumentasjon som for «SSRN-id3440802.pdf» og «Bilag 29.pdf», at denne enkodingen er mer vanlig på nyere Windows operativsystemer, kan dette tyde på at filen har blitt opprettet senere enn metadataene utgir seg for.*

Se Vedlegg 23: Innholdsanalyse «Bilag 30.doc» og Vedlegg 24: Font-oversikt «Bilag 30.doc» og «Bilag 31.doc» for detaljer.

3.2.4.10 Bilag 31

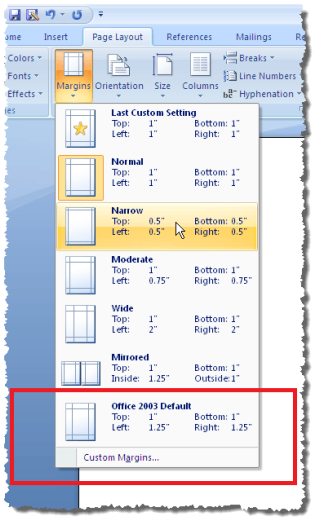
Beskrivelse: «Bilag 31.doc» er et tekstdokument med det som ser ut til å være notater.

Papirstørrelse: Filens papirstørrelse er A4, som er standardstørrelse for alle land utenom USA og Canada, jf. ISO-216.

Marger: Filens marger er 1 inch (2,54 cm) for både topp, bunn, venstre og høyre. Standard margstørrelse for MS Office Word 97-2003 dokumenter er 1-tommes (2,54 cm) marger for topp og bunn, og 1,25 tommes (3,17cm) marger for venstre og høyre. Standardstørrelsen for marger for MS Word fra 2007 og senere er 1-tommes marger for topp, bunn, venstre og høyre.

Skjermbilde fra helpdeskgeek.com [41]:

If you need to temporarily change the margins to values other than the default values for a specific document in Word, click the **Page Layout** tab and select a set of predefined margin values from the drop-down menu.



[41] Hentet fra https://helpdeskgeek.com/office-tips/change-the-default-margins-used-in-new-word-documents/, 30.11.2021

Fonter: Undersøker man enkodingen til fontene i PDF-versjonen av filen er det brukt fonter i Arial og Arial-Bold med Type: TrueType og Encoding: Ansi.

3.2.4.11 Bilag 32

Beskrivelse: «Bilag 32.doc» fremstår å være en versjon av et whitepaper.

Papirstørrelse: Papirstørrelsen til denne filen er «Standard letter» (8,5 x 11 tommer), som er likt papirstørrelsen til «bitcoin.pdf». Konverterer man et dokument fra PDF til 97-2003 Word-filformat ved hjelp av Adobe Acrobat Pro 17 vil Word-dokumentet arve papirstørrelsen til PDF-dokumentet. Standard papirstørrelse for et 97-2003 Word-dokument opprettet på en maskin som er konfigurert i et land som bruker ISO-216 vil være A4, mens det i Nord-Amerika vil være «Standard Letter».

Fonter: Denne filen bruker fontene Arial, Cambria, CourierNew, LucidaSansUnicode og TimesNewRoman. Denne filen bruker Cambria-fonten på overskrifter der «bitcoin.pdf» bruker Century Schoolbook. «bitcoin.pdf» bruker heller ikke fonten LucidaSansUnicode.

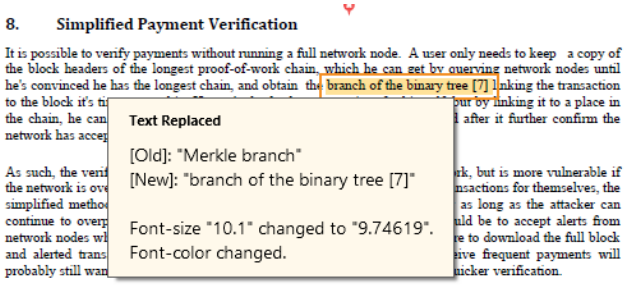
For denne filen observerer vi at det kun brukes fonter med font-type TrueType og Ansi-enkoding.

Se *Vedlegg 28: Font-oversikt fra tekst: Bilag 27, 28 & 32»* for alle fonter brukt i denne filen.

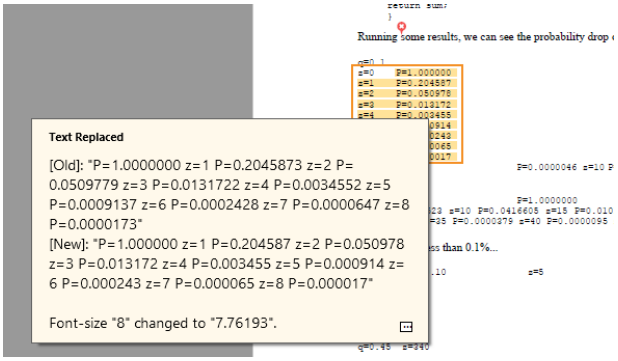
Innholdsanalyse 1: For lettere å sammenlikne teksten i filen med «bitcoin.pdf» har vi brukt bilaget «2018-11-16_B032_-_Bitcoin__A_Peer-to-Peer_Electronic_Cash_System__C._Wright__sist_endret.pdf.pdf, som er en PDF-versjon av «Bilag 32.doc».

Vi observerer at teksten i dokumentene er generelt like, med noen unntak. Denne filen har ingen forfatterinformasjon, sidetall eller figurer. I tillegg har «Bilag 32.doc» gjennomgående ingen bruk av tabulator ved nye avsnitt.

Eksempel på forskjeller i tekst ser vi fra side 4 i «Bilag 32.doc» der det brukes uttrykket «branch of binary tree», hvor det i «bitcoin.pdf» brukes «Merkle branch».



På side 6 i «Bilag 32.doc» ser vi også at tallformatteringen er forskjellig fra «bitcoin.pdf» ved at p-verdiene har et siffer mindre for z-verdiene 0 til 8, men har 7 siffer for z=9 og z=10:



I avsnittet for referanser på filens siste side mangler det også en URL for referanse nummer 6.

Se forøvrig *Vedlegg 26: Innholdsanalyse «Bilag 32.doc»* for alle forskjeller i dokumentene.

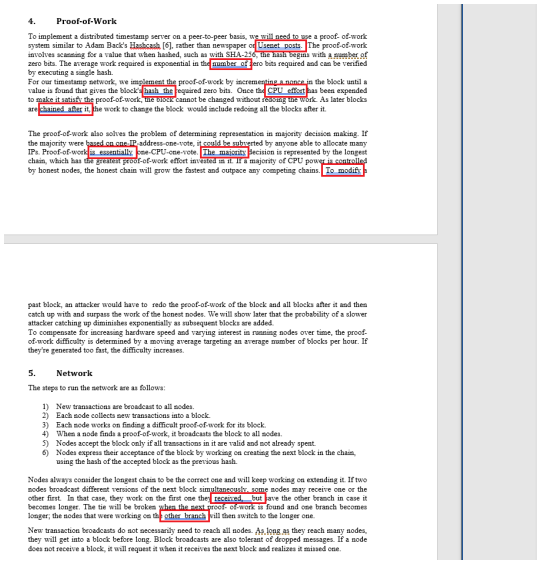
Innholdsanalyse 2:

KPMG har også sammenliknet «Bilag 32.doc» med «bitcoin-export-word.docx», som er en referansefil hvor KPMG har lastet ned «bitcoin.pdf» fra bitcoin.org og eksportert dokumentet til et Word-dokument via Adobe Acrobat Pro.

KPMG observerer at det er marg-forskjeller mellom filene.

Med unntak for kapittel 1 og 12, observerer KPMG at «Bilag 32.doc» og «bitcoin-export-word.docx» har 32 eksakt like doble mellomrom fra kapittel 2-11. Et eksempel på dette er vist under.

Kapittel 4 og 5, «Bilag 32.doc» til venstre, «bitcoin-export-word.doc» til høyre:



4. Proof-of-Work

To implement a distributed timestamp server on a peer-to-peer basis, we will need to use a proof- of-work system similar to Adam Back's Hashcash [6], rather than newspaper or Usenet posts. The proof-of-work involves scanning for a value that when hashed, such as with SHA-256, the hash begins with a number of zero bits. The average work required is exponential in the number of zero bits required and can be verified by executing a single hash.

For our timestamp network, we implement the proof-of-work by incrementing a nonce in the block until a value is found that gives the block's hash the required zero bits. Once the CPU effort has been expended to make it satisfy the proof-of-work, the block cannot be changed without redoing the work. As later blocks are chained after it, the work to change the block would include redoing all the blocks after it.

The proof-of-work also solves the problem of determining representation in majority decision making. If the majority were based on one-IP-address-one-vote, it could be subverted by anyone able to allocate many IPs. Proof-of-work is essentially one-CPU-one-vote. The majority decision is represented by the longest chain, which has the greatest proof-of-work effort invested in it. If a majority of CPU power is controlled by honest nodes, the honest chain will grow the fastest and outpace any competing chains. To modify a past block, an attacker would have to redo the proof-of-work of the block and all blocks after it and then catch up with and surpass the work of the honest nodes. We will show later that the probability of a slower attacker catching up diminishes exponentially as subsequent blocks are added.

To compensate for increasing hardware speed and varying interest in running nodes over time, the proof-of-work difficulty is determined by a moving average targeting an average number of blocks per hour. If they're generated too fast, the difficulty increases.

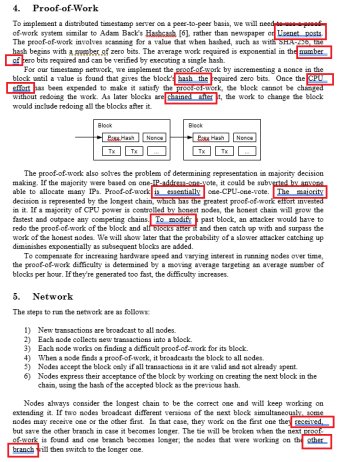
5. Network

The steps to run the network are as follows:

1) New transactions are broadcast to all nodes.
2) Each node collects new transactions into a block.
3) Each node works on finding a difficult proof-of-work for its block.
4) When a node finds a proof-of-work, it broadcasts the block to all nodes.
5) Nodes accept the block only if all transactions in it are valid and not already spent.
6) Nodes express their acceptance of the block by working on creating the next block in the chain, using the hash of the accepted block as the previous hash.

Nodes always consider the longest chain to be the correct one and will keep working on extending it. If two nodes broadcast different versions of the next block simultaneously, some nodes may receive one or the other first. In that case, they work on the first one they received, but save the other branch in case it becomes longer. The tie will be broken when the next proof- of-work is found and one branch becomes longer; the nodes that were working on the other branch will then switch to the longer one.

New transaction broadcasts do not necessarily need to reach all nodes. As long as they reach many nodes, they will get into a block before long. Block broadcasts are also tolerant of dropped messages. If a node does not receive a block, it will request it when it receives the next block and realizes it missed one.



4. Proof-of-Work

To implement a distributed timestamp server on a peer-to-peer basis, we will need to use a proof- of-work system similar to Adam Back's Hashcash [6], rather than newspaper or Usenet posts. The proof-of-work involves scanning for a value that when hashed, such as with SHA-256, the hash begins with a number of zero bits. The average work required is exponential in the number of zero bits required and can be verified by executing a single hash.

For our timestamp network, we implement the proof-of-work by incrementing a nonce in the block until a value is found that gives the block's hash the required zero bits. Once the CPU effort has been expended to make it satisfy the proof-of-work, the block cannot be changed without redoing the work. As later blocks are chained after it, the work to change the block would include redoing all the blocks after it.

Block
Prev Hash
Nonce
Tx
Tx
...

Block
Prev Hash
Nonce
Tx
Tx
...

The proof-of-work also solves the problem of determining representation in majority decision making. If the majority were based on one-IP-address-one-vote, it could be subverted by anyone able to allocate many IPs. Proof-of-work is essentially one-CPU-one-vote. The majority decision is represented by the longest chain, which has the greatest proof-of-work effort invested in it. If a majority of CPU power is controlled by honest nodes, the honest chain will grow the fastest and outpace any competing chains. To modify a past block, an attacker would have to redo the proof-of-work of the block and all blocks after it and then catch up with and surpass the work of the honest nodes. We will show later that the probability of a slower attacker catching up diminishes exponentially as subsequent blocks are added.

To compensate for increasing hardware speed and varying interest in running nodes over time, the proof-of-work difficulty is determined by a moving average targeting an average number of blocks per hour. If they're generated too fast, the difficulty increases.

5. Network

The steps to run the network are as follows:

1) New transactions are broadcast to all nodes.
2) Each node collects new transactions into a block.
3) Each node works on finding a difficult proof-of-work for its block.
4) When a node finds a proof-of-work, it broadcasts the block to all nodes.
5) Nodes accept the block only if all transactions in it are valid and not already spent.
6) Nodes express their acceptance of the block by working on creating the next block in the chain, using the hash of the accepted block as the previous hash.

Nodes always consider the longest chain to be the correct one and will keep working on extending it. If two nodes broadcast different versions of the next block simultaneously, some nodes may receive one or the other first. In that case, they work on the first one they received, but save the other branch in case it becomes longer. The tie will be broken when the next proof- of-work is found and one branch becomes longer; the nodes that were working on the other branch will then switch to the longer one.

3

KPMG anser det derfor som sannsynlig at «Bilag 32.doc» er eksportert med utgangspunkt i filen «bitcoin.pdf» til Word. For alle forskjeller vennligst se vedlegg *4.26.3*.

Marger: «Bilag 32.doc» innehar 1-tommes (2,54 cm) marger for topp og bunn, og 1,25 tommes (3,17cm) marger for venstre og høyre, som er standard for et MS Word 97-2003 dokument [42] .

3.2.4.12 Bilag 33

Beskrivelse: «Bilag 33.TIF» er et flytskjema

Innhold: Teksten i flytskjemaet er for lite til å tyde hva det skulle være, både manuelt og via OCR-behandling.

3.2.4.13 Bilag 34

Beskrivelse: «Bilag 34.TIF» er et flytskjema

Innhold: Teksten i flytskjemaet er for lite til å tyde hva det skulle være, både manuelt og via OCR-behandling.

Filen inneholder dog en datotidsstempling «9/06/2008 11:24». Denne sammenfaller med datotidsstemplingen i «Bilag 29.pdf». Dette virker da å være en TIF-fil av det samme innholdet fra den filen. For «Bilag 29.pdf» er det mye som tyder på at metadata virker å være endret i ettertid.

[42] Hentet fra https://helpdeskgeek.com/office-tips/change-the-default-margins-used-in-new-word-documents/, 30.11.2021.

3.2.4.14 Bilag 35 – 48

Disse bilagene er kategorisert som *Kildekode*. For observasjoner av disse, se seksjon **3.3 Kildekode**.

3.2.4.15 Bilag 49 og 74

Beskrivelse: Bilagene «Bilag 49.ICO» og «Bilag 74.ICO» er to bildefiler som viser Bitcoin ikonet. Begge bilag inneholder nøyaktig samme fil.

Observasjon: Basert på fil-header i disse filene har disse BMP filformat. (Fil header er 42 4D hex), ICO-filer er en kontainerfil som inneholder bildefil i BMP eller PNG-format. De to ikonfilene synes å mangle en transparent bakgrunn, og vises med sort omriss. Ved lagring i BMP-format vil eventuelt transparente områder forsvinne da BMP-formatet ikke støtter dette.

Bilag 49 og 74 (BMP-fil)		`$ file Bilag\ 49.bmp` `Bilag 49.bmp: PC bitmap, Windows 3.x format, 48 x 48 x 32, resolution 4724 x 4724 px/m, cbSize 9270, bits offset 54`
Bitcoin v0.1.0 (ICO-fil)		`$ file bitcoin.ico` `bitcoin.ico: MS Windows icon resource - 6 icons, 16x16, 8 bits/pixel, 32x32, 8 bits/pixel`

3.2.4.16 Bilag 50 og 57

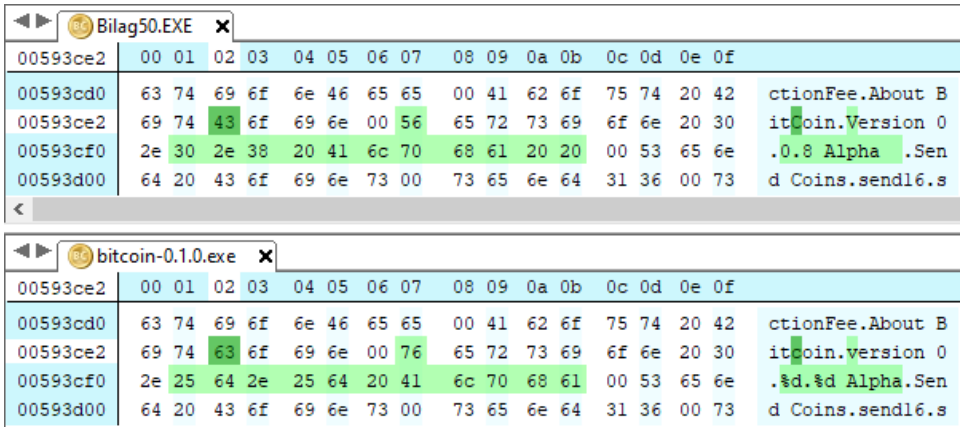
Beskrivelse: «Bilag 50.EXE» og «Bilag 57.EXE» inneholder nøyaktig samme fil basert på deres MD5-hash og vi vil videre kun referere til Bilag 50. Dette er den ferdigkompilerte programvaren (EXE), som vi har sammenlignet med programvaren som ble publisert i 0.1.0-releasen:

Observasjon: «Bilag 50.EXE» og «Bitcoin.exe» fra 0.1.0-releasen er nesten identiske, det er kun noen små forskjeller i deler med lesbar tekst.

En byggeprosess resulterer sjeldent i nøyaktig samme filstørrelse, selv på samme datamaskin og samme programvare som blir brukt, men både «Bilag 50.EXE og «Bitcoin.exe» har nøyaktig samme størrelse (6440960 bytes).

Forskjellen mellom «Bilag 50.EXE» og «Bitcoin.exe» på de delene som inneholder lesbar tekst, er at «Bilag 50.exe» inneholder ekstra mellomrom på slutten av enkelte linjer.

Dette tyder på på at dette er endringer som ble gjort med bakgrunn i den offentlig tilgjengelige kildekoden, hvor mellomrom ble brukt for å beholde den opprinnelige lengden på fila. En binærfil som er resultat av kompilert kode, ville i stedet med høyst sannsynlighet hatt null-termineringstegn på samme sted der «Bitcoin 50.EXE» har mellomrom, samt hatt en annen filstørrelse.



Utsnittet over viser enda en ytterligere indikasjon på at «Bilag 50.exe» er redigert med utgangspunkt i offentlig tilgjengelig kildekode. Kildekoden (filen «ui.cpp» i Bilag 81) sier følgende:

```
m_staticTextVersion->SetLabel(strprintf("version
0.%d.%d Alpha", VERSION/100, VERSION%100));
```

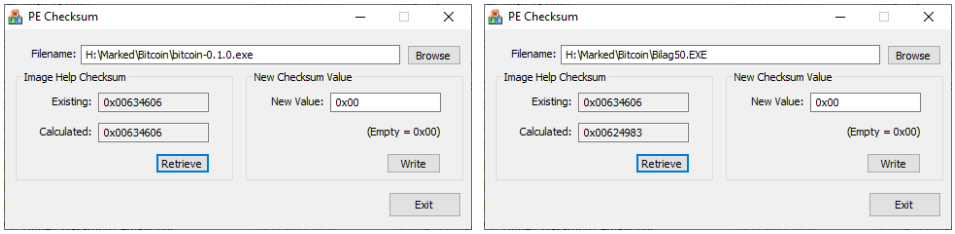
Dette forteller kompilatoren at versjonsnummeret skal erstattes dynamisk, under kjøringen av programmet. Man kan se i «Bitcoin.exe» fra 0.1.0-releasen at dette stemmer overens med kildekoden. «Bilag 50.EXE» har derimot en endring der versjonsnummeret er hardkodet («0.0.8»), noe som er inkonsekvent med egen kildekode (Bilag 81), der det fremgår at versjoneringen skal være dynamisk («0.%d.%d»).

Ovennevnte kildekodeutsnitt bekrefter samtidig at det ikke skulle vært to ekstra mellomrom etter ordet «Alpha», som er enda en indikasjon på at «Bilag 50.EXE» har blitt endret med utgangspunkt i «Bitcoin.exe» fra 0.1.0-releasen, og ikke er et resultat av en kompilering fra underliggende kildekode.

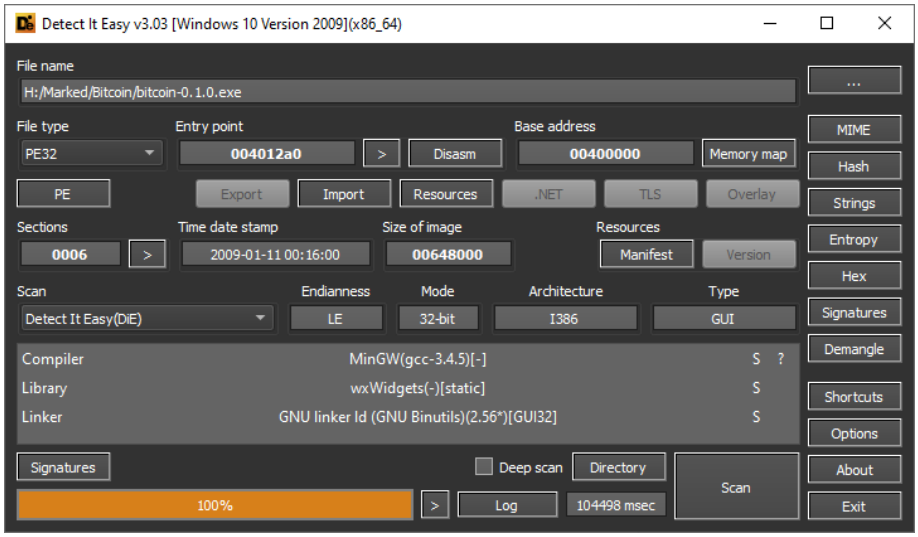
Ved nærmere analyse av metadata av «Bilag 50.EXE» og «Bitcoin.exe», ser man at byggetidspunkt og kontrollsum (checksum) er like for begge filer. Både byggetidspunktet og kontrollsummen ville høyst sannsynlig vært annerledes mellom filene dersom de var resultat av to ulike kompileringer. Dette er ikke tilfelle her – både kontrollsummen og byggetidspunkt er samme. *Byggetidspunkt: 2009-01-11 00:16:00*.

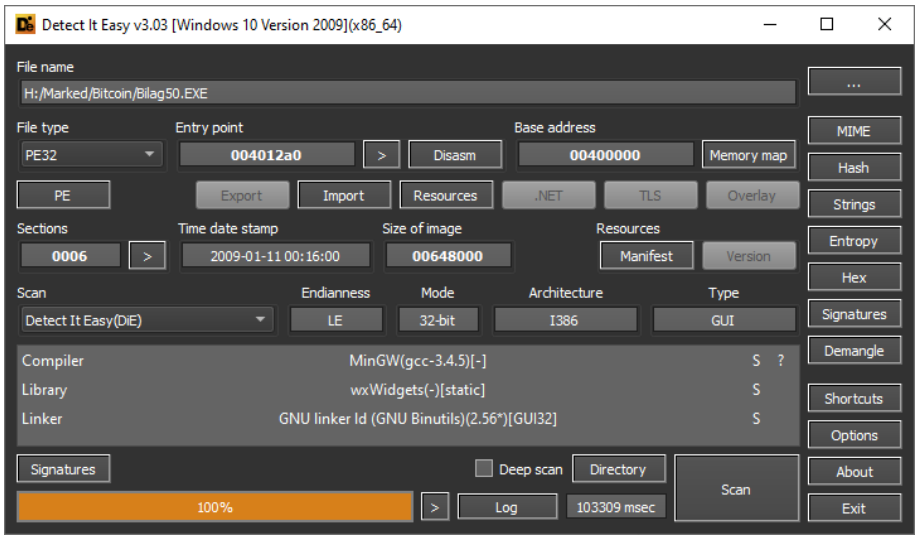
Kontrollsummen brukes for å verifisere at en fil ikke har blitt endret, for eksempel som et resultat av en feil i overføringen. I den offisielle «Bitcoin.exe» er kontrollsummen 0x00634606 (prefiks «0x» betyr at det er heksadesimal notasjon). «Bilag 50.EXE» inneholder den samme kontrollsummen i selve binærfilen, men hadde denne vært et resultat av kompilering av kildekode, med alle endringer som befinner seg ellers, ville den kalkulerte kontrollsummen vært 0x00624983.

Kontrollsum fra hhv. Bitcoin v0.1.0 og «Bilag 50.EXE»:



Byggetidspunkt og annen metadata fra hhv. Bitcoin v0.1.0 og «Bilag 50.EXE»:





Kommentar: Basert på observasjonene anser vi det som sannsynlig at «Bilag 50.EXE» og «bitcoin.exe» har vært helt identiske, der «Bilag 50.

EXE» har blitt endret i ettertid ved hjelp av en binærfil-editor, til forskjell fra å være resultat fra to forskjellige kompileringer.

3.2.4.17 Bilag 51 – 53

Disse bilagene er kategorisert som *Kildekode*. For observasjoner av disse, se seksjon **3.3 Kildekode**.

3.2.4.18 Bilag 54 og 90

Beskrivelse: «Bilag 54.LOG» og «Bilag 90.LOG» er loggfiler som skal være resultater av to kjøringer av programvaren.

Observasjon: Både «Bilag 54.LOG» og «Bilag 90.LOG» inneholder kjøringer som viser feil ved kall av GetMyExternalIP() i loggen. Dette kallet brukes for at klienten, ved hjelp av en ekstern tjeneste (whatismyip.com) skal finne ut sin egen eksterne IP-adresse.

I «Bilag 54.LOG» og «Bilag 90.LOG» sier teksten at klienten sender et kall mot 72.233.89.199 som feiler. Det kan være mange grunner til at denne feiler, men feilmeldingen sammenfaller i stor grad med en feil som først ble rapportert i oktober 2013 på github.com[43].

Kommentar: Det er ikke mulig å si noe definitivt om datoen på denne loggfilen. Feilmeldingen og hvordan den overlapper med feilrapporten på github.com kan indikere at kjøringen som resulterte i «Bilag54.LOG» og «Bilag 90.LOG» ble gjort etter oktober 2013. Feilen fra koden og feilrapporten fra github.com er illustrert i skjermbildene under.

Skjermbilde fra kjøring fra «Bilag 54.LOG»:

```
addrLocalHost = 127.0.0.1:8333
bound to addrLocalHost = 127.0.0.1:8333

ERROR: GetMyExternalIP() : connection to 72.233.89.199:80 failed
```

[43] Hentet fra https://github.com/bitcoin/bitcoin/issues/3058 22.11.2021.

Feilen som først ble rapportert i oktober 2013 på github.com [44]:

3.2.4.19 Bilag 91

Beskrivelse: «Bilag 91.PDF» er en scan av et håndskrevet notat.

Andre observasjoner: Når innholdet fra det håndskrevne filen er produsert er ikke mulig å fastslå.

[44] Hentet fra https://github.com/bitcoin/bitcoin/issues/3058 22.11.2021.

3.3 Kildekode

Se *Vedlegg 29: Bilag med kildekode* for en oversikt over bilagene med kildekode og hvordan de refererer seg til den offisielle utgivelsen v0.1.0.

3.3.1 Endringer i copyright

Bilag dette gjelder

35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 75, 76, 77, 78, 81, 82, 83, 84, 85, 86, 87, 88 og 89.

Observasjon

En rekke av kildekodefilene har endret navn i copyright-kommentar øverst i filen. Tegnsettingen i copyrighten er ikke konsistent, f.eks. er det enkelte plasser skrevet punktum eller komma etter benevnelsen «Dr».

Bitcoin v0.1.0[45]:

```
// Copyright (c) 2009 Satoshi Nakamoto
```

Bitcoin v0.0.8, oversendt i prosesskriv fra Wikborg Rein:

```
// Copyright (c) 2008 Dr Craig Wright
// Copyright (c) 2008 Dr, Craig Wright
// Copyright (c) 2008 Dr. Craig Wright
```

Kommentar

Kildekodefiler skrevet i de fleste programmeringsspråk, inkludert C++ som er benyttet i dette tilfellet, kan inneholde *kommentarer*. Kommentar markeres ved å starte linjen med to skråstreker `//`. Kommentarer benyttes bla. til å dokumentere funksjonalitet, skrive notater, copyright, el. direkte i kildekoden.

I prosessen når kildekode gjøres om til kjørbar kode (kompilering), vil kompilatoren se bort fra tekst med syntaks som tilsier at det er en kommentar. Kommentarer påvirker følgelig ikke funksjonaliteten til det endelige programmet.

Disse bilagene kan endres som rene tekstfiler og dette kan enkelt gjøres når som helst. Det er ikke mulig å si noe om filenes opphav eller datoer basert utelukkende på at copyrighten øverst i filen er endret.

3.3.2 Bugfix mapAddress.count()

Bilag dette gjelder

39, 63

[45] Hentet fra https://web.archive.org/web/20130524071850/http://www.zorinaq.com/pub/bitcoin-0.1.0.rar 17.11.21

Observasjon

I en privat mailutveksling mellom Satoshi Nakamoto og Hal Finney den 10.01.2009 ble Nakamoto gjort oppmerksom av Finney, som testet programvaren, at programvaren i visse tilfeller kunne krasje[46].

Nakamoto fant ut av problemet, lagde en feilretting, og erstattet den publiserte kildekoden med ny versjon kort tid etter. Det ble derimot ikke laget en ny versjonsnummerering for denne nye versjonen, og den ble hetende versjon 0.1.0, selv om det i realiteten var å anse som en senere versjon.

Det har ikke lyktes KPMG å finne en kilde som fortsatt har den opprinnelige versjonen av 0.1.0-koden uten feilrettingen.

Feilrettingen besto i å fjerne kall mot metoden mapAddresses.count().

From: **Satoshi Nakamoto** <satoshi@vistomail.com>
Date: Sat, Jan 10, 2009 at 2:59 PM
Subject: Re: Crash in bitcoin 0.1.0
To: hal.finney@gmail.com

I was temporarily able to reproduce the bug and narrowed it down to the "mapAddresses.count" in the following code. It was absolutely the last piece of code to go in and mainly only got tested with the MSVC build. It's not essential and I'm inclined to turn off optimization and delete the section of code until I figure out what's going on.

I'm attaching a dbg exe you can try that deletes the line of code and turns off optimization. I'm not able to reproduce it anymore at the moment.

irc.cpp:

```
if (pszName[0] == 'u')
{
    CAddress addr;
    if (DecodeAddress(pszName, addr))
    {
        CAddrDB addrdb;
        if (AddAddress(addrdb, addr))
            printf("new ");
        else
        {
            // make it try connecting sooner
            CRITICAL_BLOCK(cs_mapAddresses)
                if (mapAddresses.count(addr.GetKey()))
                    mapAddresses[addr.GetKey()].nLastFailed = 0;
        }
        addr.print();
    }
    else
    {
        printf("decode failed\n");
```

Kommentar

Versjonen av koden fremlagt i «Bilag 39» inneholder heller ikke dette problematiske kallet, i stedet matcher den koden som er har ligget tilgjengelig i offentlighet etter at feilrettingen var implementert.

Dersom forfatteren av «Bilag 39» hevder at kildekoden i dette bilaget er basert på en versjon av kildekoden datert før den offentlig tilgjengelige versjonen av 0.1.0-kildekoden,

[46] Hentet fra https://chainbulletin.com/satoshi-used-timestamps-for-early-bitcoin-source-code-versioning/ og https://online.wsj.com/public/resources/documents/finneynakamotoemails.pdf

så må forfatteren av «Bilag 39» selv ha implementert en identisk feilretting som den som finnes i den offisielle 0.1.0-versjonen.

3.3.3 Bitcoin-adresse-eksempel

Bilag dette gjelder

84

Observasjon

En analyse av adressen:

`12STD5BhabrNpx56pWuC6wctxz3Qf2gdD7`

viser at den bruker såkalt komprimert offentlig nøkkel[47] med verdien:

`0347b872d0eff3c69523f6eebfc95b8144e6d115cee1b834ec36c576549bdbfa0f`

Kommentar

Komprimert offentlig nøkkel var ikke støttet av Bitcoin-klienten før versjon 0.6 som først ble publisert i 2012[48]:

[47] Hentet fra https://iancoleman.io/bitcoin-key-compression/ 23.11.2021.

[48] Hentet fra https://bitcointalk.org/index.php?topic=74737.0 23.11.2021.

```
NEW FEATURES SINCE BITCOIN VERSION 0.5
--------------------------------------

Initial network synchronization should be much faster
(one or two hours on a typical machine instead of ten or more
hours).

Backup Wallet menu option.

Bitcoin-Qt can display and save QR codes for sending
and receiving addresses.

New context menu on addresses to copy/edit/delete them.

New Sign Message dialog that allows you to prove that you
own a bitcoin address by creating a digital
signature.

New wallets created with this version will
use 33-byte 'compressed' public keys instead of
65-byte public keys, resulting in smaller
transactions and less traffic on the bitcoin
network. The shorter keys are already supported
by the network but wallet.dat files containing
short keys are not compatible with earlier
versions of Bitcoin-Qt/bitcoind.
```

3.3.4 Ingen forskjell mellom Bitcoin v0.1.0 og v0.0.8

Bilag dette gjelder

51, 52, 53, og 83

Observasjon

Det er ingen forskjeller mellom tilsvarende filer i de to versjonene.

3.3.5 Release notes

Bilag

47 og 75

Observasjon

Filen er tekstfil med versjonsmerknader knyttet til utgivelsen av kildekoden.

Endret tekst fra `BitCoin v0.01 ALPHA` til `BitCoin Version 0.0.8 Alpha` på linje 1

Endret tekst fra `MinGW GCC (v3.4.5)` til `MinGW GCC (v3.4.4)` på linje 13

Kommentar

KPMG kan ikke se at det finnes en versjon 3.4.4 av MinGW GCC på de offisielle sidene til MinGW GCC[49].

[49] Hentet fra https://sourceforge.net/projects/mingw/files/MinGW/Base/gcc/Version3/

3.3.6 Utkommentert hashGensisBlock

Bilag

67

Observasjon

Bilag 67 fra prosesskrivet fra WR er en fil med kildekode som inneholder en utkommentert linje. Denne utkommenterte linjen finnes ikke i den tilsvarende filen i den offisielle 0.1.0 versjonen på Satoshi Nakamoto Institute.

```
// const uint256
hashGenesisBlock("0x000006b15d1327d67e971d1de9116bd60a3a01556c91b6ebaa416ebc0cfa
a646");
```

Kommentar

Det er ikke mulig å si noe om hvorfor denne linjen med utkommentert kode er lagt til i filen som er en del av prosesskrivet til WR.

3.3.7 Endringer i mellomrom

Bilag

79 og 80

Observasjon

I bilag 79 og 80 er det kun funnet forskjeller i form av et ekstra mellomrom på slutten av linje 5. Dette har ingen betydning for hverken menneskelig eller maskinmessig tolkning av kildekoden.

Kommentar

Det er ikke mulig å si noe om hvorfor dette mellomrommet er lagt til i filene.

4 Vedlegg

4.1 Vedlegg 1: Filoversikt mottatte filer

Filoversikt: Mottatte filer				
Fil-beskrivelse	**Filnavn**	**MD5-hash**	**Fil-type**	**Til analyse**
WR	2015-06-29_B001_-_Summary_of_Agreed_Terms__med_kommentarer_.pdf.pdf	c7205a5e9b58a965ee6e43dd8810ea7b	PDF	
WR	Bilag 1.pdf	6605c3c928d6ea6f2d64b77d63575432	PDF	
WR	2015-06-29_B002_-_Revidert_Summary_of_Agreed_Terms.pdf.pdf	481d3fcf07d176fe8f9784782d3df4b6	PDF	
WR	Bilag 2.pdf	987e817a314ce0213b72be7d1fec84cf	PDF	
WR	2016-01-07_B003_-_IP_Assignment_Deed__Craig_Wright__Ncrypt_Holdings_Ltd_.PDF.pdf	ffa9a1adfa866e38536cc79614adb442	PDF	
WR	Bilag 3.PDF	ce9cd08a8b9e52e8cae299385b25fddd	PDF	
WR	2016-01-07_B004_-_IP_Assignment_Deed__Panopticrypt_Pty_Ltd__Ncrypt_Holdings_Ltd_.PDF.pdf	28fd81e85d695b2f130b7a8c831fa0a3	PDF	
WR	Bilag 4.PDF	361921e6197f562898c2f8ab0de2e6aa	PDF	
WR	2016-01-07_B005_-_IP_Assignment_Deed__NCrypt_Holdings_Ltd._mv._.PDF.pdf	faa314dee314d83003798c9524ca8e12	PDF	
WR	Bilag 5.PDF	ee0e69bf746ccfdebca8df1a94856662	PDF	
WR	2016-02-17_B006_-_Life_Story_Rights_and_Services_Agreement.PDF.pdf	dcdfd638e08b3d3e4add0206f6bfdeb0	PDF	
WR	Bilag 6.PDF	10436800d0df30b88b22fa371fc75b01	PDF	
WR	2016-08-22_B007_-_Deed_of_Amendment_to_Life_Story_Rights_and_Services_Agreement.PDF.pdf	fd54b580ba60e060c34fcbf0d47dc143	PDF	
WR	Bilag 7.PDF	048390c6e92fbc48488555b7b68bce49	PDF	
WR	2020-05-04_B008_-_Termination__Release_and_Assignment-back_Agreement.PDF.pdf	0631db9b4fb4fe84ccc42d8cbc37b240	PDF	
WR	Bilag 8.PDF	008dd67489985dc3e229cc60185aee37	PDF	
WR	2015-11-21_B009_-_E-postkorr._mellom_Craig_Wright_og_Robert_MacGregor_mfl.__20.-21.11.2015.pdf.pdf	24052abcfad7003143e070d7a5853b1d	PDF	
WR	Bilag 9.pdf	e921f7d218a830a4306a7c6e11b5d23b	PDF	
WR	2015-12-03_B010_-_E-postkorr._mellom_Craig_Wright_og_Robert_MacGregor_mfl.__02.-03.12.2015.pdf.pdf	40bb0aa8df5e92f56bd0c454239c56dc	PDF	

WR	Bilag 10.pdf	6d9bd6f36fc38dd9d143d0e8f9e bddde	PDF	
WR	2015-12-05_B011_-_E-postkorr._mellom_Craig_Wright_og_Robert_MacGregor_mfl..pdf.pdf	531c8b0afa3155593841bb8976 6780bf	PDF	
WR	Bilag 11.pdf	6624bc679853f11258e0b19dad b6a438	PDF	
WR	2016-03-11_B012_-_NDA_for_Jon_Matonis__usignert_.pdf.pdf	b395c00a229495c63218280a6 8a7b18b	PDF	
WR	Bilag 12.pdf	3ba577b211d4c4123bd2f0d175 4fb58c	PDF	
WR	2016-03-16_B013_-_NDA_for_Gavin_Andresen.pdf.pdf	e9b273b13a09319072b923c66 0018182	PDF	
WR	Bilag 13.pdf	e2fa2ea905ff7546f54dfdfe94e4 2423	PDF	
WR	2016-04-25_B014_-_NDA_for_Nicolas_T._Courtois.pdf.pdf	3bfdd86f3d50c694d12dda67c0 7a2dc0	PDF	
WR	Bilag 14.pdf	7a35dc592c9eddc46931dd39f5 63c98d	PDF	
WR	2020-02-26_B015_-_E-poster_vist_til_ifm._Gavin_Andresens_vitneforklaring.PDF.pdf	8ba179f7567c16fff250bca614e e0c66	PDF	
WR	Bilag 15.PDF	25c9e3b6402bf4a67dab8cb952 b3b067	PDF	
WR	2016-05-03_B016_-_E-post_fra_MacGregor_til_Wright_og_Ramona.pdf.pdf	3c2cdf613b18ea40ee592be1c9 d682e0	PDF	
WR	Bilag 16.pdf	f789915d4ff2f82e13002717b60 4ca69	PDF	
WR	2016-05-03_B017_-_Utkast_til_bloggpost.pdf.pdf	0a3df921bfc886846885a03af16 dc7a5	PDF	
WR	Bilag 17.docx	9c5f07240532d6a79180dd7c8d c3c4fe	docx	
WR	2016-05-03_B018_-_E-post_til_MacGregor_med_svar_fra_Ramona.pdf.pdf	77ca136e230e897fffd041857f7 cfb6c	PDF	
WR	Bilag 18.pdf	727588bfb0cbb8dedf12accb74b f1b6c	PDF	
WR	2014-03-06_B019_-_E-post_fra_Wright_til_Ira_Kleiman.pdf.pdf	506a3ea86878690978b4f19b62 e0ff5f	PDF	X
WR	Bilag 19.pdf	795a79a8d6526531c9f42fc56a a31665	PDF	X
WR	B020_-_Tidlig_utkast_Hviteboken__Bitcoin_A_Peer-to-Peer_Electronic_Cash_System__Satoshi_Nakamoto.PDF.pdf	f1afdef88b87384d2ca1f84c008 bd232	PDF	X
WR	Bilag 20.PDF	8408769f3720b76e24f54b0571 6dcbc5	PDF	X
WR	B021_-_2007-08-00_-_H_ndskrevet_f_rsteutkast_av_Hviteboken__Craig_Wright.PDF.pdf	559d10089cde15126f352dbb4e 091bb2	PDF	X
WR	Bilag 21.PDF	d060e9d552fee80da80fb6831e a44ce2	PDF	X
WR	B022_-_2007-08-00_-_BDO_m_tereferattidsplan_for_lanseringen_av_bitcoin.PDF.pdf	5d45dcca0a645532cdfead835a 188707	PDF	X
WR	Bilag 22.PDF	71ce1d59067745a639ee40062 53539ef	PDF	X
WR	B023_-_2008-00-00_-_TimeChain__utskrift_kode_med_kommentarer_fra_Craig_Wright.PDF.pdf	5db4c29d56172372c6e27efbd2 076a72	PDF	X

WR	Bilag 23.PDF	75813b22cc54ca604cb703ba9858d509	PDF	X
WR	B024_-_2008-00-00_-_Opprinnelig_kode_for_bitcoin__utskrift_med_komm._og_utregn._fra_Craig_Wright.PDF.pdf	54f8c9fc40f9d68664c3df1977129c28	PDF	X
WR	Bilag 24.PDF	f48e8edcb2c9137ab0595a5bc004fb17	PDF	X
WR	B025_-_H_ndskrevne_notater_om_utviklingen_av_bitcoin__Craig_Wright.PDF.pdf	de4358c4d674d8dcdb93237dc0d7989f	PDF	X
WR	Bilag 25.PDF	ce6335fffb3845f2f9956d16bd3428d0	PDF	X
WR	2008-01-05_B026_-_Artikkel__K._Suichi__med_h_ndskrevne_komm._Craig_Wright.PDF.pdf	761519fa98e901fce021f60ba1a00cdd	PDF	X
WR	Bilag 26.PDF	457785a0691cee2915ef34d932d9ce2d	PDF	X
WR	2008-05-06_B027_-_TimeCoin_Peer-to-Peer_Electronic_Cash_System__Dr._Craig_S._Wright.pdf.pdf	e08fb65af3e6e56942a741528688adc8	PDF	X
WR	Bilag 27.ODT	7f8befdd723ff197f461f7fba21b32fb	odt	X
WR	2008-05-21_B028_-_Bitcoin_A_peer-to-Peer_Electronic_Cash_System__Dr._Craig_Wright__sist_endret.PDF.pdf	36a7d2f2b7e6c0d9d163b0d10f9fad26	PDF	X
WR	Bilag 28.PDF	42fa5efcd463e895c4a1aa7f5612f02f	PDF	X
WR	2008-06-09_B029_-_Flytskjema__visualisering_av_f_rste_bitcoin_kildekode.PDF.pdf	4b1fc23b29b2d0c4efbf28757077e6d1	PDF	X
WR	Bilag 29.PDF	ec9a2e200159fb5a06b54cf1ba286647	PDF	X
WR	2008-10-23_B030_-_Bitcoin__Craig_Wright__sist_endret.pdf.pdf	2c0b7c9c1f3c18d97c66968cdc9544a5	PDF	X
WR	Bilag 30.DOC	080a98f16eeeda8defbd15e2b4fac7f2	DOC	X
WR	2008-10-23_B031_-_Bitcoin__law___Craig_Wright__sist_endret.pdf.pdf	f6f6751bc746560c2de45d361da60d80	PDF	X
WR	Bilag 31.DOC	4128db5c270060e1464b9d516b3de8ea	DOC	X
WR	2018-11-16_B032_-_Bitcoin__A_Peer-to-Peer_Electronic_Cash_System__C._Wright__sist_endret.pdf.pdf	47025ba1845a6bc0cee383dc7080152b	PDF	X
WR	Bilag 32.DOC	b4fe862a3dc51011f1a4bfe0c53159e9	DOC	X
WR	2008-09-06_B033_-_Flytskjema__visualisering_kildekode.pdf.pdf	c7e966469066b3aaa696422240d7fb63	PDF	X
WR	Bilag 33.TIF	5dfa87ac73f3061f3e92a865de7a1f36	TIF	X
WR	2008-09-06_B034_-_Flytskjema__visualisering_kildekode.pdf.pdf	9f16ddfa44adaa8206e91d8e3159ccc3	PDF	X
WR	Bilag 34.TIF	49d03aa0f320de2e9711e5a3986195d6	TIF	X
WR	2009-01-04_B035_-_Kode_oppfin_lans._org._bitcoin_v.0.0.8__opphavsrett_Dr.Wright2008__kl_12.38_.pdf.pdf	045ec6333251044a53c496cc4007947d	PDF	X
WR	Bilag 35.H	5df318a9d211865b848bc73982811dfe	H	X

WR	2009-01-04_B036_-_Opprinnelig_bitcoin-kode_v._0.0.8__opphavsrett_Dr._Craig_Wright__kl_12.39_.pdf.pdf	172b2220d35a18bf271eb14f4c4780a4	PDF	X
WR	Bilag 36.H	81a5dac7266e4df4b9efdfbbf632348c	H	X
WR	2009-01-04_B037_-_Opprinnelig_bitcoin-kode_v._0.0.8__opphavsrett_Dr._Craig_Wright__kl_12.39_.pdf.pdf	feec02808dc8e5e8987f13fc59d2e1bc	PDF	X
WR	Bilag 37.CPP	fda6deb31982fc6fe90364235f4b929f	CPP	X
WR	2009-01-04_B038_-_Opprinnelig_bitcoin-kode_v._0.0.8__opphavsrett_Dr._Craig_Wright__kl_12.39_.pdf.pdf	181dbfb5a738df6640c44125ad0bbef0	PDF	X
WR	Bilag 38.H	e22f1e01e357a09b5240c21967a20148	H	X
WR	2009-01-04_B039_-_Opprinnelig_bitcoin-kode_v._0.0.8__opphavsrett_Dr._Craig_Wright__kl_12.39_.pdf.pdf	0e391a1b1b0c938b67c73766edb8907d	PDF	X
WR	Bilag 39.CPP	edb64a52a206f9644e83264b86549e0f	CPP	X
WR	2009-01-04_B040_-_Kode_oppfin_lans._org._bitcoin_v.0.0.8__opphavsrett_Dr.Wright2008__kl_12.39_.pdf.pdf	cb354c2e3a3c6528b6da2d7c3d93a4eb	PDF	X
WR	Bilag 40.CPP	cb918b80c35e833ea31c5afc7165cf37	CPP	X
WR	2009-01-04_B041_-_Kode_oppfin_lans._org._bitcoin_v.0.0.8__opphavsrett_Dr.Wright2008__kl_12.39_.pdf.pdf	d0a1484e643341d682bc2bf4897d168a	PDF	X
WR	Bilag 41.H	b8600343c17b6c6f4c1247490263de9d	H	X
WR	2009-01-04_B042_-_Kode_oppfin_lans._org._bitcoin_v.0.0.8__opphavsrett_Dr.Wright2008__kl_12.39_.pdf.pdf	6e91c6e7e5b9d2cd85a16f815335bed8	PDF	X
WR	Bilag 42.CPP	cffb88b2e2dffeb39f865ca295c9a958	CPP	X
WR	2009-01-04_B043_-_Kode_oppfin_lans._org._bitcoin_v.0.0.8__opphavsrett_Dr.Wright2008__kl_12.40_.pdf.pdf	79bbfdb3888a51d8e3f82a6fa833cef8	PDF	X
WR	Bilag 43.H	6764552ddf17d971a407d93dd3d11765	H	X
WR	2009-01-04_B044_-_Opprinnelig_bitcoin-kode_v._0.0.8__opphavsrett_Dr._C.Wright__kl_12.40_.pdf.pdf	b791cf184947cf29efede7f98c52e984	PDF	X
WR	Bilag 44.H	09975c5ef9c501c5cc5b678309b3aabe	H	X
WR	2009-01-04_B045_-_Kode_oppfin_lans._org._bitcoin_v.0.0.8__opphavsrett_Dr.Wright2008__kl_12.41_.pdf.pdf	97e3ab98e0a346d9b953bb9ca2b339c2	PDF	X
WR	Bilag 45.CPP	f8206d5539423d293d6803c9e4e1e205	CPP	X
WR	2009-01-04_B046_-_Kode_oppfin_lans._org._bitcoin_v.0.0.8__opphavsrett_Dr.Wright2008__kl_12.41_.pdf.pdf	7e4be62b9b00c7744c65c17a0def3978	PDF	X
WR	Bilag 46.H	320a381365f08208c1f21abee2604c21	H	X
WR	2009-01-04_B047_-_Alpha-versjon_0.0.8_bitcoin-kode__opphavsrett_Dr._Craig_Wright_2008__kl_12.42_.pdf.pdf	430cfa2c4ab395d99c79e89d8fe4eacd	PDF	X
WR	Bilag 47.TXT	a60bf9597c82b9faba7af0522cd1074d	TXT	X

WR	2009-01-04_B048_-_Oppr._bitcoin-kode_v._0.0.8__opphavsrett_Dr._Craig_Wright_2008__kl_12.46_.pdf.pdf	faf020b11254fbaa70e0d42781325aa7	PDF	X
WR	Bilag 48.FBP	83a868e0fb64ad6e0861465e86b323e8	FBP	X
WR	2009-01-04_B049_-_Tidlig_bitcoin-logo__kl_12.53_.pdf.pdf	69edbb8dcefdfd8190a63559bf3512cc	PDF	X
WR	Bilag 49.ICO	05f220f02ae09f167ef6f44ba77faf6b	ICO	X
WR	2009-01-04_B050_-_Original_bitcoin_v._0.0.8_bitcoin-programvare__kl_13.08_.pdf.pdf	3a9925996d77e572cf224a91a02dd466	PDF	X
WR	Bilag 50.EXE	e5e1190c5237c0ecbc77c7c25a86b1ef	EXE	X
WR	2009-01-07_B051_-_S.Nakamoto_meld_kode_fil-lisens_oppsett_bitcoin__opphavsrett_S.Nakamoto_2009.pdf.pdf	78f911046acc56f1e0218b08fc67ad8e	PDF	X
WR	Bilag 51.TXT	55c076538b7dd357549feefff1e2bef1	TXT	X
WR	2009-01-07_B052_-_S.Nakamoto_meld_kode_om_oppsett___koding_bitcoin__opphavsrett_S._Nakamoto_2009.pdf.pdf	8ef457e5b4e6a97d730be2c37bc1969f	PDF	X
WR	Bilag 52.H	4cbf8991ed978e0444a17bde6cd6c675	H	X
WR	2009-01-07_B053_-_Satoshi_Nakamoto_melding_kode__opphavsrett_Satoshi_Nakamoto_2009.pdf.pdf	71e683844210ba4bc6d97ec7fed18606	PDF	X
WR	Bilag 53.H	22823c178af18695224ec162babf057a	H	X
WR	2009-01-10_B054_-_Debug_log-fil_som_viser_fors_k_p____kj_re_bitcoin-programvaren__kl_07.38_.pdf.pdf	42f86777ed0a6fecb3d1416b42a3c061	PDF	X
WR	Bilag 54.LOG	3a42a21a889a84a5dabaaf51a6a2d992	LOG	X
WR	2009-01-02_B055_-_Memo___Company_Formation___fra_Craig_Wright.pdf.pdf	3fe60c97bdef15948d8e699feb11f7f9	PDF	X
WR	Bilag 55.PDF	6127f21e473afc118670ffce825f3f50	PDF	X
WR	2014-10-11_B056_-_E-post__RE_C01n_Pty_Ltd_ATO_Review___fra_C._Wright_til_J._Aitken_og_R._Watts.pdf.pdf	74613e9328b8412a955b4150cb76492b	PDF	X
WR	Bilag 56.mht	96c18ac5650d67fbfee4c677f071741d	mht	X
WR	image002.jpg	021ccc824df5f367bc40ae35598b3fe5	jpg	X
WR	image004.png	fe363d0dd0c8c1d4be2fa0a9a716d141	png	X
WR	image005.png	cda9c2089426dec9c691116081e5f6ab	png	X
WR	image006.png	5c9c932f2f92903e057697165fc24813	png	X
WR	image007.png	8a68752b8ce2e95d8e47ccc32d92df6b	png	X
WR	image008.png	480e7728cce7eb271d76f1fa1a8805dc	png	X
WR	image010.png	e787b0106f2ce285d4af772e2189d550	png	X

WR	image011.png	3401ee39515d125e4d5829e0b9fdd197	png	X
WR	image012.png	585549d425ef5ee6c58d467c3ea782ee	png	X
WR	2009-01-04_B057_-_Original_bitcoin_v._0.0.8_bitcoin-programvare__kl_13.08_.pdf.pdf	49d33bf6c7fceb20450245b728af49f9	PDF	X
WR	Bilag 57.EXE	e5e1190c5237c0ecbc77c7c25a86b1ef	EXE	X
WR	2009-01-04_B058_-_Craig_Wright_melding_kode__opphavsrett_Dr._Craig_Wright_2008__kl_12.38_.pdf.pdf	1e8dbfa05505942f90da0eaa2ae7d33e	PDF	X
WR	Bilag 58.H	5df318a9d211865b848bc73982811dfe	H	X
WR	2009-01-04_B059_-_Craig_Wright_kode__opphavsrett_Dr._Craig_Wright_2008__kl_12.39_.pdf.pdf	8fff278a95c9abb0f5a7a782ade57778	PDF	X
WR	Bilag 59.H	81a5dac7266e4df4b9efdfbbf632348c	H	X
WR	2009-01-04_B060_-_Craig_Wright_kode__opphavsrett_Dr._Craig_Wright_2008__kl_12.39_.pdf.pdf	68e8e980e82f959be28c55b8e39d4fd8	PDF	X
WR	Bilag 60.CPP	fda6deb31982fc6fe90364235f4b929f	CPP	X
WR	2009-01-04_B061_-_Craig_Wright_kode__opphavsrett_Dr._Craig_Wright_2008__kl_12.39_.pdf.pdf	0ed3c1581cc5ad23d380a90615f1b8d7	PDF	X
WR	Bilag 61.H	e22f1e01e357a09b5240c21967a20148	H	X
WR	2009-01-04_B062_-_Craig_Wright_kode__opphavsrett_Dr._Craig_Wright_2008__kl_12.39_.pdf.pdf	06750a8e8df2a1c70f3354229d653b40	PDF	X
WR	Bilag 62.H	f8d90a225df4e149a7045be978dd552d	H	X
WR	2009-01-04_B063_-_Craig_Wright_kode__opphavsrett_Dr._Craig_Wright_2008__kl_12.39_.pdf.pdf	513e90d34bd9caff440ade6f17a90021	PDF	X
WR	Bilag 63.CPP	edb64a52a206f9644e83264b86549e0f	CPP	X
WR	2009-01-04_B064_-_Craig_Wright_kode__opphavsrett_Dr._Craig_Wright_2008__kl_12.40_.pdf.pdf	0bba93c606559511640b6b012055f363	PDF	X
WR	Bilag 64.H	a3ce3772bf46797c994f3f962f837fa8	H	X
WR	2009-01-04_B065_-_Craig_Wright_kode__opphavsrett_Dr._Craig_Wright_2008__kl_12.40_.pdf.pdf	856b1a0d7d762b6eb654010b21b576ac	PDF	X
WR	Bilag 65.H	09975c5ef9c501c5cc5b678309b3aabe	H	X
WR	2009-01-04_B066_-_Melding_om_opphavsrett__Dr._Craig_Wright_2008__kl_12.40_.pdf.pdf	a2f5abf7ea85f3f198783263b241e5e5	PDF	X
WR	Bilag 66.TXT	59fe067fdbc5e7a36b73ac990f3188ce	TXT	X
WR	2009-01-04_B067_-_Craig_Wright_kode__opphavsrett_Dr._Craig_Wright_2008__kl_13.06_.pdf.pdf	a725a3c8e5f027f7f9b6c6d7ceb e84c3	PDF	X
WR	Bilag 67.CPP	d59d60d8a7a1c64fa66f4063a6242ee0	CPP	X

WR	2009-01-04_B068_-_Craig_Wright_kode__opphavsrett_Dr._Craig_Wright_2008__kl_12.48_.pdf.pdf	c3433198c39170e4a3c3140f8cdc8781	PDF	X
WR	Bilag 68.H	210750d83eb9e296735317c15d1dbea1	H	X
WR	2009-01-04_B069_-_Craig_Wright_kode__opphavsrett_Dr._Craig_Wright_2008__kl_12.47_.pdf.pdf	62e409074bd7db0297aa3cb1b3db706f	PDF	X
WR	Bilag 69.VC	fa0b0e812abc3732dfacc62e150e362d	VC	X
WR	2009-01-04_B070_-_Craig_Wright_kode__opphavsrett_Dr._Craig_Wright_2008__kl_12.41_.pdf.pdf	ccfc6c2d175cedf3399f687d850be0ee	PDF	X
WR	Bilag 70.CPP	f8206d5539423d293d6803c9e4e1e205	CPP	X
WR	2009-01-04_B071_-_Craig_Wright_kode__opphavsrett_Dr._Craig_Wright_2008__kl_12.41_.pdf.pdf	4163df32b991e0213f9b327188c58282	PDF	X
WR	Bilag 71.H	320a381365f08208c1f21abee2604c21	H	X
WR	2009-01-04_B072_-_Craig_Wright_kode__opphavsrett_Dr._Craig_Wright_2008__kl_12.39_.pdf.pdf	5963dd692cac78bf413120d97b2af1fe	PDF	X
WR	Bilag 72.CPP	cb918b80c35e833ea31c5afc7165cf37	CPP	X
WR	2009-01-04_B073_-_Craig_Wright_kode__opphavsrett_Dr._Craig_Wright_2008__kl_12.41_.pdf.pdf	ea44217ff9383bb24d2b90434b8b11a4	PDF	X
WR	Bilag 73.H	c2eb671fcccf57a9e6f086a073340c2d	H	X
WR	2009-01-04_B074_-_Tidlig_bitcoin-logo__kl_12.53_.pdf.pdf	f453868a07ea1027a1aa006a77bb14f5	PDF	X
WR	Bilag 74.ICO	05f220f02ae09f167ef6f44ba77faf6b	ICO	X
WR	2009-01-04_B075_-_Alpha-versjon_0.0.8_bitcoin-kode__opphavsrett_Dr._Craig_Wright_2008__kl_12.42_.pdf.pdf	e9e98553c2f2127dfc55a391a4cb9432	PDF	X
WR	Bilag 75.TXT	a60bf9597c82b9faba7af0522cd1074d	TXT	X
WR	2009-01-04_B076_-_Craig_Wright_kode__opphavsrett_Dr._Craig_Wright_2008__kl_12.42_.pdf.pdf	ddf7785662153a1af8e32099063ef6c4	PDF	X
WR	Bilag 76.CPP	24fdfbf4fef7abad9b95d7bb25130583	CPP	X
WR	2009-01-04_B077_-_Craig_Wright_kode__opphavsrett_Dr._Craig_Wright_2008__kl_12.43_.pdf.pdf	1a93a2665092d67afb1c77e253573078	PDF	X
WR	Bilag 77.H	f380209e112c1fbe09d22b64c150d336	H	X
WR	2009-01-04_B078_-_Craig_Wright_kode__opphavsrett_Dr._Craig_Wright_2008__kl_12.43_.pdf.pdf	125af3e4d5943ea3e07b01aea7152ca1	PDF	X
WR	Bilag 78.H	3dbf36f8f26c3f517ec9da966f8c99b4	H	X
WR	2009-01-04_B079_-_Kode__SHA_routines_Crypto____versjon_5.5.2__24._september_2007__kl_12.47_.pdf.pdf	9827a975a0eba196383a68f5ca629aee	PDF	X
WR	Bilag 79.CPP	991d004344465a39b19bc5c2f7ea5134	CPP	X

WR	2009-01-04_B080_-_Kode__SHA_routines_Crypto____versjon_5.5.2__24._september_2007__kl_12.47_.pdf.pdf	b3f411c279dc9c4dc2ecafe73abf67ab	PDF	X
WR	Bilag 80.H	00b705e470c58a782035f646b0c01678	H	X
WR	2009-01-04_B081_-_Craig_Wright_kode__opphavsrett_Dr._Craig_Wright_2008__kl_12.43_.pdf.pdf	3e22df5642c4b4fe2c7d43f053b97675	PDF	X
WR	bilag 81.CPP	f364b805fb17f685b97c58b584b0e5a4	CPP	X
WR	2009-01-04_B082_-_Craig_Wright_kode__opphavsrett_Dr._Craig_Wright_2008__kl_12.43_.pdf.pdf	a2ff36207c5e17d7169e8a7011114eb0	PDF	X
WR	Bilag 83.RC	f0119f58b4ab8d298afb1d9a67cda050	RC	X
WR	2009-01-04_B083_-_Craig_Wright_kode__opphavsrett_Dr._Craig_Wright_2008__kl_12.47_.pdf.pdf	d90c6f9a46276c96d9a55d1d9b83e024	PDF	X
WR	Bilag 84.CPP	1701f0a502a043a7a57cf06752d14604	CPP	X
WR	2009-01-04_B084_-_Craig_Wright_kode__opphavsrett_Dr._Craig_Wright_2008__kl_12.45_.pdf.pdf	e3f8904278fa422bef217d9cca874a18	PDF	X
WR	Bilag 85.H	327ea0f1133e84f98ad9df072a96e289	H	X
WR	2009-01-04_B085_-_Craig_Wright_kode__opphavsrett_Dr._Craig_Wright_2008__kl_12.42_.pdf.pdf	06fbbfcb9cfec1b031ea748eb35272c7	PDF	X
WR	Bilag 86.H	6764552ddf17d971a407d93dd3d11765	H	X
WR	2009-01-04_B086_-_Craig_Wright_kode__opphavsrett_Dr._Craig_Wright_2008__kl_12.40_.pdf.pdf	e1c98276618659def942a1032cf11cff	PDF	X
WR	2009-01-04_B087_-_Craig_Wright_kode__opphavsrett_Dr._Craig_Wright_2008__kl_12.46_.pdf.pdf	257c7971b8e71b14d1e6c25bb617f8f3	PDF	X
WR	Bilag 88.CPP	cffb88b2e2dffeb39f865ca295c9a958	CPP	X
WR	2009-01-04_B088_-_Craig_Wright_kode__opphavsrett_Dr._Craig_Wright_2008__kl_12.39_.pdf.pdf	2b1316887500e02492543f01f723720d	PDF	X
WR	Bilag 89.H	b8600343c17b6c6f4c1247490263de9d	H	X
WR	2009-01-04_B089_-_Craig_Wright_kode__opphavsrett_Dr._Craig_Wright_2008__kl_12.39_.pdf.pdf	df07ad2082839cd01eccbc496089f489	PDF	X
WR	2009-01-04_B090_-_Kode__bitcoin-wallet__kl_13.25_.pdf.pdf	9104174cbc07fcb03516f4e920aa1127	PDF	X
WR	Bilag 90.LOG	6ef348eefdc4f168567cc0ba82d60e00	LOG	X
WR	B091_-_H_ndskrevet_notat___Time_Chain___Craig_Wright.pdf.pdf	331401881725549e70eccf6582f132a9	PDF	X
WR	Bilag 91.PDF	44f48ba4dff491a4faca33a7fa47902b	PDF	X
WR	2021-08-27_2021-08-27_Prosesskriv - Google translation.docx	30f623e04c8033ae90bee6b57fb89731	docx	
WR	2021-08-27_2021-08-27_Prosesskriv.pdf.pdf	0b74047dcbbaadd41843851d0a474c30	PDF	

WR	71016231-2021-08-27_2021-08-27_Prosesskriv - Semantix translation.docx	8a1e3595e27a96cda86dd2ea9c19a033	docx	
WR	Prosesskriv fra WR.zip	4a26a008411b868af0cff7b570c036b6	mp4	
SVW	2021-07-02_Prosesskriv_SVW.pdf	fce7ae37e43e84ae65e5dbb2e25e7052	PDF	
SVW	B01, Foruminnlegg pÃ¥ bitcointalk.org datert 6. juli 2010 (hentet fra https___archive.md_ee2v3#selection-465.0-465.58).pdf	888970241bca061ea4a91d286500bd65	PDF	
SVW	B02, Innlegg pÃ¥ medium.com av Craig Wright datert 22. mars 2019 .pdf	e31421baeb12fc0e84f83e8a7b2e6a2a	PDF	
SVW	B03, Blogginnlegg Â«SSH Port ForwardingÂ» datert 10. januar 2012, arkivert 3. oktober 2015.pdf	40a1524998e9b7a002e31ae15a66655c	PDF	
SVW	B04, Blogginnlegg Â«SSH Port ForwardingÂ» datert 10. januar 2012, arkivert 23. november 2015.pdf	01d80daed0ff27d8155ea0204651ae8f	PDF	
SVW	B05, Summary of Agreed Terms, datert 29. juni 2015.pdf	b63714bf7076cfd01634b7f99d3aec33	PDF	
SVW	B06, Artikkel fra Wired Â«Is Bitcoin's Creator this Unknown Australian Genius_ Probably Not (Updated)Â» datert (opprinnelig) 8. desember 2015 (oppdatert 30. april 2019).pdf	ae136da539f5bf970969453d2d25914a	PDF	
SVW	B07, Artikkel fra Gizmodo Â«The Mystery of Craig Wright and Bitcoin Isn't Solved YetÂ» datert (opprinnelig) 11. desember 2015.pdf	de1ec536278f15f2b20a64a1c78f1c94	PDF	
SVW	B08, Artikkel fra Wired.com datert 2. mai 2016 .pdf	b54045bd0de80a64c9cc5851ebb92487	PDF	
SVW	B09, Boken Â«The Satoshi AffairÂ» av Andrew Oâ€™Hagan datert 30. juni 2016 .pdf	b52523c2d1aa471b8cfa83aac4890fda	PDF	
SVW	B10, Artikkel fra BBC datert 2. mai 2016 .pdf	c5e664b17c0024f72c09944f63a5c619	PDF	
SVW	B11, Artikkel fra The Economist datert 2. mai 2016 .pdf	2b27c731b0e820f6f62ec744d366e030	PDF	
SVW	B12, Artikkel fra BBC datert 2. mai 2016.pdf	b9893d7f13556e9feda9be5c5ca3386f	PDF	
SVW	B13, Innlegg Â«The Craig Wright May 2016 Signing Sessions Debacle, In Full ContextÂ» datert 30. juni 2021 .pdf	9a7509c564da98e133a10e20f1e4e62d	PDF	
SVW	B14, Deposition of Gavin A. Andresen datert 26. februar 2020 .pdf	35f7307510d3ea767f346bb9b30ec83e	PDF	
SVW	B15, Blogginnlegg av Dr. Nicolas Courtois Â«Is Satoshi Nakamoto Back_Â» datert 2. mai 2016.pdf	a27f7c8f836e09af5d5a895f8852a003	PDF	
SVW	B16, Blogginnlegg av Dr. Nicolas Courtois Â«A Short Human-Verifiable Proof that Craig Wright has Cheated the PressÂ» datert 2. mai 2016.pdf	0d750b0b48ad7b46d2d7518f6aada80a	PDF	
SVW	B17, Utdrag fra kapittel 10 Â«Crypto CrazinessÂ» i Rory Cellan-Jonesâ€™ bok Â«Always OnÂ» (2021).pdf	7ef0833dd5b7b8903d81b01a25f80f11	PDF	
SVW	B18, Wrights originale post Â«I'm SorryÂ», datert 5. mai 2016.pdf	857ff8a7b6325c76915b0bf63e319e11	PDF	
SVW	B19, Artikkel fra nettstedet silliconangle.com Â«Craig Wright faces criminal charges and serious jail time in UK after claiming to be Bitcoin's founder Satoshi NakamotoÂ» datert 3. mai 2016, arkivert 4. mai 2016 .pdf	08586d23e855c1ae468282f1df12d35d	PDF	
SVW	B20, Nettsiden silliconangle.com, arkivert 9. mai 2016.pdf	e87d6102f37270b0b3e824c493b09838	PDF	
SVW	B21, Artikkel fra nettstedet Bitcoinist.com, Â«UK Law Enforcement Sources Hint At Impending Crait Wright ArrestÂ» .pdf	9375cb98e4fdcc97aff1a853e2325076	PDF	

SVW	B22, Nettsiden silliconangle.com, arkivert 6. mars 2016.pdf	6816c41b2eef5b02c55bef3103e9f016	PDF	
SVW	B23, Nettsiden silliconangle.com, arkivert 6. mars 2016 .pdf	2f6841e0783fdd410d8bde8785565658	PDF	
SVW	B24, Â«Amended complaintÂ» (Kleiman v. Wright) datert 14. mai 2018 .pdf	cd88896fb08b8d7dcc75e410a6a9e88a	PDF	
SVW	B25, Â«Deed of LoanÂ» datert 23. oktober 2012 .pdf	74a53bea0c5c3b1d1da38fff922632de	PDF	
SVW	B26, Melding med signatur datert 16. mai 2019 .pdf	25331f7e7f3a08bf3fed2ffe2ded20fd	PDF	
SVW	B27, Artikkel Â«The Reason for LawÂ» datert 8. april 2019 .pdf	00df9bc4629ccb9c9aff02ff1ac55b71	PDF	
SVW	B28, Artikkel Â«Why code must not be lawÂ» av Craig Wright, datert 18. mai 2019 .pdf	4677f3647873e364d87b20d873c7e895	PDF	
SVW	B29,Â«Notice of complianceÂ» datert 14. januar 2020 .pdf	a5cc9fade2a0f4270566e50cdbc5c30a	PDF	
SVW	B30, Â«Plaintiffs' Omnibus Sanctions MotionÂ» datert 21. mai 2020 .pdf	f710062f945868ef2df0071fc3e418d8	PDF	
SVW	B31, Â«Plaintiffs' Omnibus Sanctions MotionÂ» Exhibit 7 inngitt 21. mai 2020 .pdf	36da7445bd579883d515e1204a800968	PDF	
SVW	B32, Â«Notice of supplemental evidenceÂ» datert 27. mai 2020 .pdf	7aff5a50c4cc3d5df0fba0d635e876bd	PDF	
SVW	B33, Epost fra Craig Wright til Dave Kleiman datert 12. mars 2008.pdf	62dac455e52479b537ddc8791b79bee7	PDF	
SVW	B34, Artikkel Â«The Strange Life and Death of Dave Kleiman, A Computer Genius Linked to Bitcoin's OriginsÂ» datert 9. Desember 2015.pdf	0d9432beb51980901a4562640b2791b6	PDF	
SVW	B35, Declaration of Craig Wright datert 8. mai 2019 .pdf	8555a3cfb9c8f91f5d40439d06120548	PDF	
SVW	B36, E-poster mellom Mike Hearn og Satoshi Nakamoto datert 25.-27. april 2009.pdf	2a1a45cf703044f72ab651d399a7973e	PDF	
SVW	B37, Twittermelding av 5. mars 2019, arkivert 7. mars 2019.pdf	976f0f63f263e26c27e37158bb44a724	PDF	
SVW	B38, Blogginnlegg Â«Why PI Laws do not impact Digital ForensicsÂ» datert 7. november 2007, arkivert 29. oktober 2013 .pdf	26f4a792dfc577e7a2096c5ff26fce92	PDF	
SVW	B39, Videoklipp fra Craig Wrights presentasjon 30. mai 2019.pdf	ef91f0ce77d1d016e3d03aad924572b3	PDF	
SVW	B40, Epost fra Satoshi Nakamoto til Wei Dai av 22. august 2008.pdf	51a1331fdf976efe2dc5fef2a440f5df	PDF	
SVW	B41, Â«Bitcoin_ A Peer-to-Peer Electronic Cash SystemÂ» av 31. oktober 2008.pdf.pdf	3f0feee9fb0ed0fd07fa748e280096fe	PDF	
SVW	B42, Â«Bitcoin_ A Peer-to-Peer Electronic Cash SystemÂ» av 24. mars 2009.pdf.pdf	d56d71ecadf2137be09d8b1d35c6c042	PDF	
SVW	B43, Skjermdump fra SSRN.com hvor dokumentet SSRN-id3440802 kan lastes ned.pdf	b925df7b13047900e80bc5c474826722	PDF	
SVW	B44, Dokumentet SSRN-id3440802.pdf_ Â«Bitcoin_ A Peer-toPeer Electronic Cash SystemÂ» datert 21. august 2008.pdf	5e210ce3003ffaed204b7b2076c2fc92	PDF	
SVW	B45, Egenskaper til dokumentet SSRN-id3440802.pdf	dcd397fd8f87e0d5bfdc18c46186da54	PDF	
SVW	B46, Metadata for SSRN-id3440802 fra metadata2go.com.pdf	0c1d64ccfc0944ee6b6ed10ac2092b47	PDF	
SVW	B47, Blogginnlegg Â«Implementing Fraud Detection using Bayesian methods in Data Sets with Benford's LawÂ» datert 26. november 2007, arkivert 2. juni 2014 .pdf	f934e8c1d9bfa57e9bc5c7de8c4d8726	PDF	

SVW	B48, Blogginnlegg Â«Implementing Fraud Detection using Bayesian methods in Data Sets with Benford's LawÂ» datert 26. november 2007, arkivert 3. juni 2015 .pdf	b346b2fe31950c9172b861893f989cb2	PDF	
SVW	B49, Blogginnlegg Â«TonightÂ» datert 26. august 2008, arkivert 2. juni 2014 .pdf	d998f792c9b62e8db9fda8aed0703611	PDF	
SVW	B50, Blogginnlegg Â«TonightÂ» datert 26. august 2008, arkivert 25. mai 2015.pdf	7ea0de29de628fc30f7e5aa635aacad1	PDF	
SVW	B51, Artikkel Â«Have We Finally the Creator of Bitcoin_â€ datert 8. desember 2015 .pdf	f99f70c59af07d23d407d9ea95d4297a	PDF	
SVW	B52, Skjermdump av blogginnlegget.pdf	b41a2658b088549120027259a83076bc	PDF	
SVW	B53, Forumdiskusjon pÃ¥ Reddit .pdf	a363df6995862605a69337dd88e6ac28	PDF	
SVW	B54, Blogginnlegg Â«IntegrysÂ» datert 12. mai 2009, arkivert 2. juni 2014.pdf	64e2a9525a0d251f06ca20bf9fe6ad59	PDF	
SVW	B55, Blogginnlegg Â«IntegrysÂ» datert 12. mai 2009, arkivert 25. mai 2015.pdf	322b5cf2eb0cc36f7ed5556f5ecd50d2	PDF	
SVW	B56, Blogginnlegg Â«Why gold_Â» datert 11. februar 2011, arkivert 25. mai 2015 .pdf	49e573ee8d8dbc4ac611abff3e77eca7	PDF	
SVW	B57, Blogginnlegg Â«Why gold_Â» datert 11. februar 2011, arkivert 22. september 2015 .pdf	f3c328cd6237f45c548d8fde881a9d46	PDF	
SVW	B58, Wikipedia â€“ Satoshi Nakamoto.pdf	3e9262ad39337c87fd50c8c3cd1be242	PDF	
SVW	B59, Wikipedia â€“ Craig Steven Wright .pdf	b1b01f3329e1dc9187befc76f45da316	PDF	
SVW	B60, Twitter-melding fra Peter Todd av 30. juni 2017 .pdf	ea5a63890035b7e6fbc6a5ebf7101365	PDF	
SVW	B61, Twitter-melding fra Samson Mow av 20. oktober 2017 .pdf	84e7bd23f97a63681d71c04f7a848ed2	PDF	
SVW	B62, Twitter-melding fra Charlie Lee av 5. april 2018 .pdf	e6bc185f2389be8eeddbc6b268344be5	PDF	
SVW	B63, Twitter-meldinger fra Wikileaks av 12. februar 2019 .pdf	4b40b85907879ad1c909975fd7e94a2e	PDF	
SVW	B64, Twitter-melding fra Riccardo Spagni av 30. mars 2019 .pdf	6e7ae274572980693a65560ecb80000c	PDF	
SVW	B65, Twitter-melding fra Eric Lombrozo av 9. april 2019 .pdf	173f329c09c473f34c7f6e74737e4d80	PDF	
SVW	B66,Â«Deposition of Jimmy NguyenÂ» datert 30. april 2020.pdf	b52b5def646a67249cdbe0ac858ee887	PDF	
SVW	B67, Innlegg av Jack Laskey og Dave Mullen-Muhr pÃ¥ unboundedcapital.com Â«Why we think Craig Wright Is Satoshi and why that matters (July 2020 update)Â» (oppdatert versjon) datert 9. juli 2020.pdf	84527a95aedd07e021bf43de60793cb1	PDF	
SVW	B68, Artikkel av Sam Williams pÃ¥ Medium.com Â«Response to Unbounded Capitalâ€™s attempt to show that Craig Wright is SatoshiÂ» datert 10. januar 2020.pdf	48d92d2c49b920795d32473592cb7bf9	PDF	
SVW	B69, Artikkel pÃ¥ Coingeek Â«What is BitcoinÂ» udatert.pdf	ce7bdf28259518336c96427ed3d7cbee	PDF	
SVW	B70, Slack-samtale postet pÃ¥ reddit.com datert 3. juni 2019.pdf	5d88872a72feeb5e2990a546de784c3b	PDF	
SVW	B71, Tweet av Alistair Milne datert 18. november 2019.pdf	0e9b215d37de892cc86bb19bc9a3eea9	PDF	
SVW	B72, Tweet av Arthur van Pelt datert 10. september 2019.pdf	9c98ba184937d0c7728b2d71455d6ca9	PDF	
SVW	Prosesskriv fra SVW.zip	a3190620346d66902771299fba2355c5	ZIP	

SVW	Screen_Recording_20210629-110038_1.mp4	f40f48e9db2ed3d2501407f80eb72dfe	xml	

4.2 Vedlegg 2: Filoversikt referansefiler

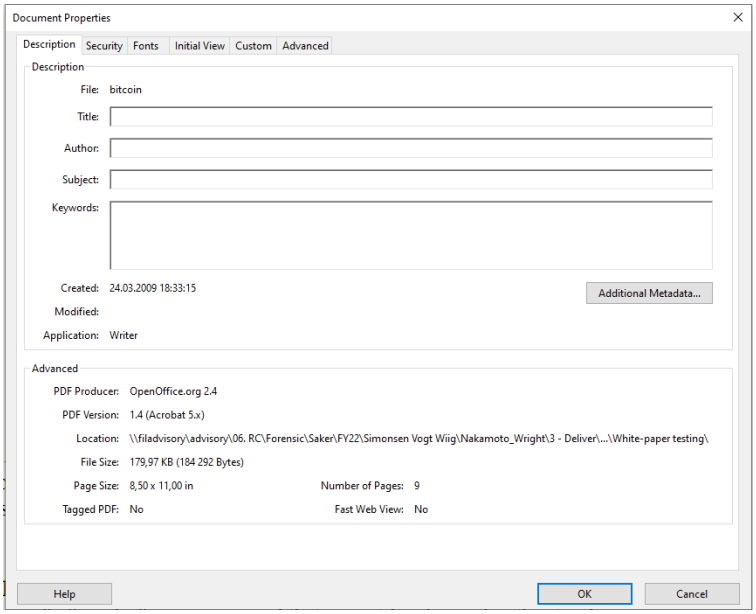
Filoversikt: Referansefiler					
Fil-beskrivelse	**Filnavn**	**MD5-hash**	**Fil-type**	**Kilde**	**Dato for nedlasting**
Referansefil	bitcoin.pdf	d56d71ecadf2137be09d8b1d35c6c042	PDF	https://bitcoin.org/bitcoin.pdf	11.11.2021 12:07
Referansefil	SSRN-id3440802.pdf	5e210ce3003ffaed204b7b2076c2fc92	PDF	https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3440802	11.11.2021 12:18
Referansefil	bitcoin-draft.pdf	b7026c5be02de23871fc1d80a49e087b	PDF	http://satoshinakamoto.me/bitcoin-draft.pdf	23.11.2021 15:31
Referansefil	bitcoin-kpmg-edit.pdf	a2b7fab4eca8f3b5838aaac2bf343f7c	PDF	Egen KPMG-redigert whitepaper	06.12.2021 09:19
Referansefil	J-stor-original-article.pdf	0fd9ea0b3134cd353238e28ecc856e13	PDF	https://www.jstor.org/stable/2383230	23.11.2021 11:08
Referansefil	ODT Testdokument.odt	3d4c15510bf69c64271eba94f4d8a65b	ODT	Eget KPMG-redigert testdokument	23.11.2021 12:51
Referansefil	PDF til Word til odt (Acrobat Reader).odt	ACB5D02A281959532E16CD1D1C8E2BD1	ODT	Eget KPMG-redigert testdokument	23.11.2021 14:46
Referansefil	1967-kato.pdf	996aa51675d2bb834f765ce2cb7eca93	PDF	https://web.archive.org/web/20160509080944/http://www.gwern.net/docs/tominaga/1967-kato.pdf	12.12.2021 19:00
Referansefil	bitcoin-export-word.docx	f8a923fb41f6a869330db1deccf992a	DOCX	Konvertert til Word fra PDF fra https://bitcoin.org/bitcoin.pdf	10.12.2021 17:34
Referansefil	Bilag28_unc.pdf	41101ac34346622111b78ea83e8eb156	PDF	Dekomprimert originalbilag	10.12.2021 13:56
Referansefil	Bilag29_unc.pdf	a7a8c838df99e6635939def1e4d88c5a	PDF	Dekomprimert originalbilag	10.12.2021 14:28
Referansefil	SSRN_unc.pdf	ecee5e6deeebb438d5e41739f1e2f89d	PDF	Dekomprimert referansefil https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3440802	10.12.2021 13:34
Referansefil	makefile	6915D8E33CD93261EB9D1272E3799EC0	TXT	https://web.archive.org/web/20130524071850/http://www.zorinaq.com/pub/bitcoin-0.1.0.rar	17.11.2021 14:39
Referansefil	addressbook16.bmp	442307926BC3FCC2118E67D7EB9D0F1D	BMP	https://web.archive.org/web/20130524071850/http://www.zorinaq.com/pub/bitcoin-0.1.0.rar	17.11.2021 14:39
Referansefil	addressbook16mask.bmp	D2D7A8E2781E93BEBE3509DE3DAF83B1	BMP	https://web.archive.org/web/20130524071850/http://www.zorinaq.com/pub/bitcoin-0.1.0.rar	17.11.2021 14:39
Referansefil	addressbook20.bmp	5A0474014AB8CEB34007A8615BD36A08	BMP	https://web.archive.org/web/20130524071850/http://www.zorinaq.com/pub/bitcoin-0.1.0.rar	17.11.2021 14:39
Referansefil	addressbook20mask.bmp	9695939B90508BBD4C9F815E5CF3623D	BMP	https://web.archive.org/web/20130524071850/http://www.zorinaq.com/pub/bitcoin-0.1.0.rar	17.11.2021 14:39
Referansefil	send16.bmp	0AA745307D8E91352C23BA425BF082D3	BMP	https://web.archive.org/web/20130524071850/http://www.zorinaq.com/pub/bitcoin-0.1.0.rar	17.11.2021 14:39
Referansefil	send16mask.bmp	241B85366C2F6CC0F244C261DF865574	BMP	https://web.archive.org/web/20130524071850/http://www.zorinaq.com/pub/bitcoin-0.1.0.rar	17.11.2021 14:39
Referansefil	send16masknoshadow.bmp	96884C7DA51CB665BD5EACD570E412C7	BMP	https://web.archive.org/web/20130524071850/http://www.zorinaq.com/pub/bitcoin-0.1.0.rar	17.11.2021 14:39
Referansefil	send20.bmp	A3D4481F61C796DEACFDF55BCF9B1C6B	BMP	https://web.archive.org/web/20130524071850/http://www.zorinaq.com/pub/bitcoin-0.1.0.rar	17.11.2021 14:39
Referansefil	send20mask.bmp	D96799FAD965ABB565A31E77CF6A203E	BMP	https://web.archive.org/web/20130524071850/http://www.zorinaq.com/pub/bitcoin-0.1.0.rar	17.11.2021 14:39
Referansefil	db.cpp	971D002AA007AB2C577FA3DB5D234767	TXT	https://web.archive.org/web/20130524071850/http://www.zorinaq.com/pub/bitcoin-0.1.0.rar	17.11.2021 14:39
Referansefil	irc.cpp	E41E8555D0B853821EC4D3AB0668E6EF	TXT	https://web.archive.org/web/20130524071850/http://www.zorinaq.com/pub/bitcoin-0.1.0.rar	17.11.2021 14:39

Referansefil	main.cpp	E04B403D08684267AD12CA71D244BFD4	TXT	https://web.archive.org/web/20130524071850/http://www.zorinaq.com/pub/bitcoin-0.1.0.rar	17.11.2021 14:39
Referansefil	market.cpp	6C9BAFB7EE6867CA9FC32F5F63A684F1	TXT	https://web.archive.org/web/20130524071850/http://www.zorinaq.com/pub/bitcoin-0.1.0.rar	17.11.2021 14:39
Referansefil	net.cpp	040A5FD6BD38B862B85B39D5743E00F4	TXT	https://web.archive.org/web/20130524071850/http://www.zorinaq.com/pub/bitcoin-0.1.0.rar	17.11.2021 14:39
Referansefil	script.cpp	768017113C0F7D8AFB6D3A7CA93E1F66	TXT	https://web.archive.org/web/20130524071850/http://www.zorinaq.com/pub/bitcoin-0.1.0.rar	17.11.2021 14:39
Referansefil	sha.cpp	4E7AF8CE43B6AFE3D3C12BDE38DCA123	TXT	https://web.archive.org/web/20130524071850/http://www.zorinaq.com/pub/bitcoin-0.1.0.rar	17.11.2021 14:39
Referansefil	ui.cpp	EBD8CB8ADB86AE3D8CA5EB86A69DB992	TXT	https://web.archive.org/web/20130524071850/http://www.zorinaq.com/pub/bitcoin-0.1.0.rar	17.11.2021 14:39
Referansefil	uibase.cpp	7A0B4E31F2AE6FEB08BB38116BCD8549	TXT	https://web.archive.org/web/20130524071850/http://www.zorinaq.com/pub/bitcoin-0.1.0.rar	17.11.2021 14:39
Referansefil	util.cpp	25A514B43A3AB03A940EE4F6153D648A	TXT	https://web.archive.org/web/20130524071850/http://www.zorinaq.com/pub/bitcoin-0.1.0.rar	17.11.2021 14:39
Referansefil	libeay32.dll	192EF960BD269B499097FB154EF2B6F8	DLL	https://web.archive.org/web/20130524071850/http://www.zorinaq.com/pub/bitcoin-0.1.0.rar	17.11.2021 14:39
Referansefil	mingwm10.dll	D6109C0E39F2D4D2FB86505159021B3F	DLL	https://web.archive.org/web/20130524071850/http://www.zorinaq.com/pub/bitcoin-0.1.0.rar	17.11.2021 14:39
Referansefil	bitcoin.exe	307AD86C412C02ABEB821AFBB900355B	EXE	https://web.archive.org/web/20130524071850/http://www.zorinaq.com/pub/bitcoin-0.1.0.rar	17.11.2021 14:39
Referansefil	uiproject.fbp	EF8A605E21C5A1C8C97B1717E1929DE7	XML	https://web.archive.org/web/20130524071850/http://www.zorinaq.com/pub/bitcoin-0.1.0.rar	17.11.2021 14:39
Referansefil	base58.h	4CBF8991ED978E0444A17BDE6CD6C675	TXT	https://web.archive.org/web/20130524071850/http://www.zorinaq.com/pub/bitcoin-0.1.0.rar	17.11.2021 14:39
Referansefil	bignum.h	22823C178AF18695224EC162BABF057A	TXT	https://web.archive.org/web/20130524071850/http://www.zorinaq.com/pub/bitcoin-0.1.0.rar	17.11.2021 14:39
Referansefil	db.h	D702E77C555B184E4AA5C49B54FF1BD2	TXT	https://web.archive.org/web/20130524071850/http://www.zorinaq.com/pub/bitcoin-0.1.0.rar	17.11.2021 14:39
Referansefil	headers.h	21336A2D1E0E78A99FEEEA04B97F8493	TXT	https://web.archive.org/web/20130524071850/http://www.zorinaq.com/pub/bitcoin-0.1.0.rar	17.11.2021 14:39
Referansefil	irc.h	47C918062348E4B2F296282624C89650	TXT	https://web.archive.org/web/20130524071850/http://www.zorinaq.com/pub/bitcoin-0.1.0.rar	17.11.2021 14:39
Referansefil	key.h	1E9AEE11EDE3B43563BCB642FABA4333	TXT	https://web.archive.org/web/20130524071850/http://www.zorinaq.com/pub/bitcoin-0.1.0.rar	17.11.2021 14:39
Referansefil	main.h	3C22E20B46744AB5185C5FCDEE39127A	TXT	https://web.archive.org/web/20130524071850/http://www.zorinaq.com/pub/bitcoin-0.1.0.rar	17.11.2021 14:39
Referansefil	market.h	3A2001CFB55CD4ECC71CC839DF95F929	TXT	https://web.archive.org/web/20130524071850/http://www.zorinaq.com/pub/bitcoin-0.1.0.rar	17.11.2021 14:39
Referansefil	net.h	2F0786BBC70A7F46C1144BB3D0ABF6CA	TXT	https://web.archive.org/web/20130524071850/http://www.zorinaq.com/pub/bitcoin-0.1.0.rar	17.11.2021 14:39
Referansefil	script.h	F4C66945D0B7C0116F6254D4E6F747DF	TXT	https://web.archive.org/web/20130524071850/http://www.zorinaq.com/pub/bitcoin-0.1.0.rar	17.11.2021 14:39
Referansefil	serialize.h	B3032E31BCABECB9EDE4198BAE4BCF95	TXT	https://web.archive.org/web/20130524071850/http://www.zorinaq.com/pub/bitcoin-0.1.0.rar	17.11.2021 14:39
Referansefil	sha.h	D18A1BC8F860C245A835D4920297FAB0	TXT	https://web.archive.org/web/20130524071850/http://www.zorinaq.com/pub/bitcoin-0.1.0.rar	17.11.2021 14:39
Referansefil	ui.h	A6D47C99189A44318354D78E2479C6D2	TXT	https://web.archive.org/web/20130524071850/http://www.zorinaq.com/pub/bitcoin-0.1.0.rar	17.11.2021 14:39
Referansefil	uibase.h	25F171C6C3646E91BA8AD4914AE65AE2	TXT	https://web.archive.org/web/20130524071850/http://www.zorinaq.com/pub/bitcoin-0.1.0.rar	17.11.2021 14:39
Referansefil	uint256.h	D436DEC4D6ECB68E812B7BEC53591C01	TXT	https://web.archive.org/web/20130524071850/http://www.zorinaq.com/pub/bitcoin-0.1.0.rar	17.11.2021 14:39
Referansefil	util.h	D6E55DA47374A67B0CF1EC69E344ADE6	TXT	https://web.archive.org/web/20130524071850/http://www.zorinaq.com/pub/bitcoin-0.1.0.rar	17.11.2021 14:39

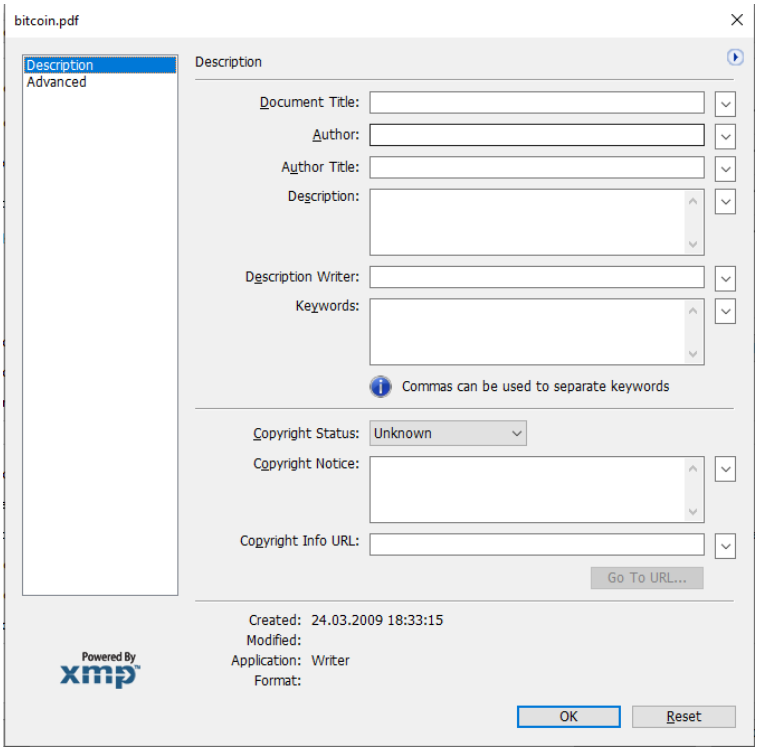
Referansefil	bitcoin.ico	27C508EBABAD3521618576EE907B5486	ICO	https://web.archive.org/web/20130524071850/http://www.zorinaq.com/pub/bitcoin-0.1.0.rar	17.11.2021 14:39
Referansefil	check.ico	A41E4872C7F291CF72B84AFE285EB9BA	ICO	https://web.archive.org/web/20130524071850/http://www.zorinaq.com/pub/bitcoin-0.1.0.rar	17.11.2021 14:39
Referansefil	ui.rc	F0119F58B4AB8D298AFB1D9A67CDA050	TXT	https://web.archive.org/web/20130524071850/http://www.zorinaq.com/pub/bitcoin-0.1.0.rar	17.11.2021 14:39
Referansefil	license.txt	96F9785634054DF9927715FEB0AC3D3B	TXT	https://web.archive.org/web/20130524071850/http://www.zorinaq.com/pub/bitcoin-0.1.0.rar	17.11.2021 14:39
Referansefil	license.txt	96F9785634054DF9927715FEB0AC3D3B	TXT	https://web.archive.org/web/20130524071850/http://www.zorinaq.com/pub/bitcoin-0.1.0.rar	17.11.2021 14:39
Referansefil	readme.txt	55C076538B7DD357549FEEFFF1E2BEF1	TXT	https://web.archive.org/web/20130524071850/http://www.zorinaq.com/pub/bitcoin-0.1.0.rar	17.11.2021 14:39
Referansefil	readme.txt	30EC76D492A532695AD074540E3CB114	TXT	https://web.archive.org/web/20130524071850/http://www.zorinaq.com/pub/bitcoin-0.1.0.rar	17.11.2021 14:39
Referansefil	makefile.vc	DF318DCEF323D9AD356C456FCF40B609	TXT	https://web.archive.org/web/20130524071850/http://www.zorinaq.com/pub/bitcoin-0.1.0.rar	17.11.2021 14:39

4.3 Vedlegg 3: Metadata «bitcoin.pdf»

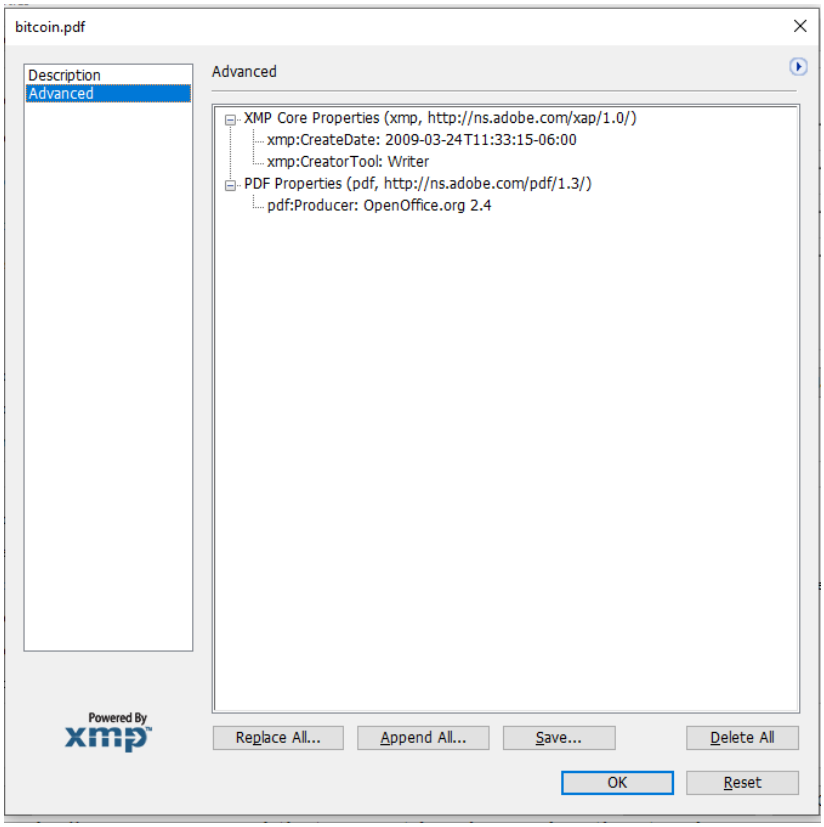
4.3.1 Adobe metadata



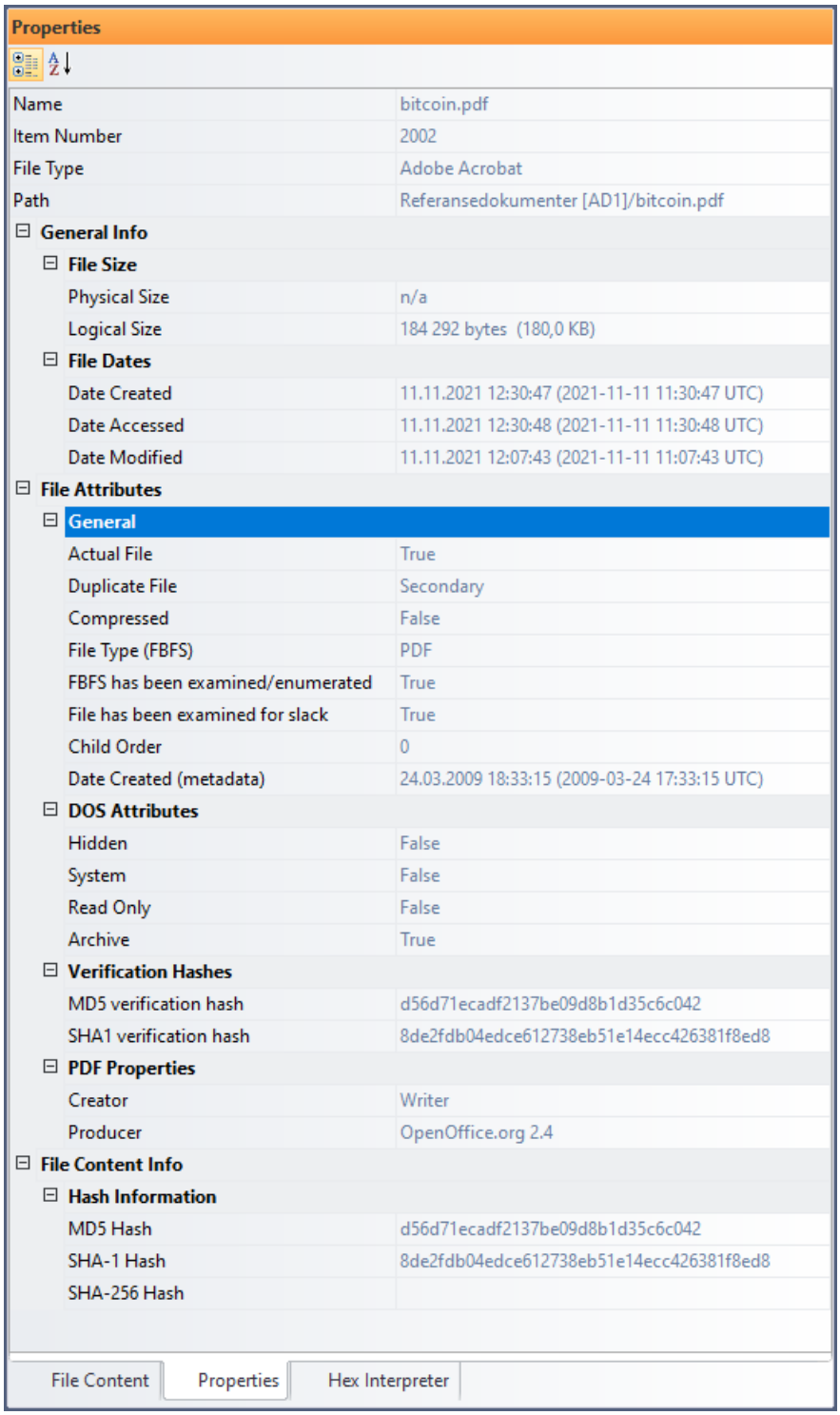
4.3.2 XMP Description



4.3.3 XMP Advanced

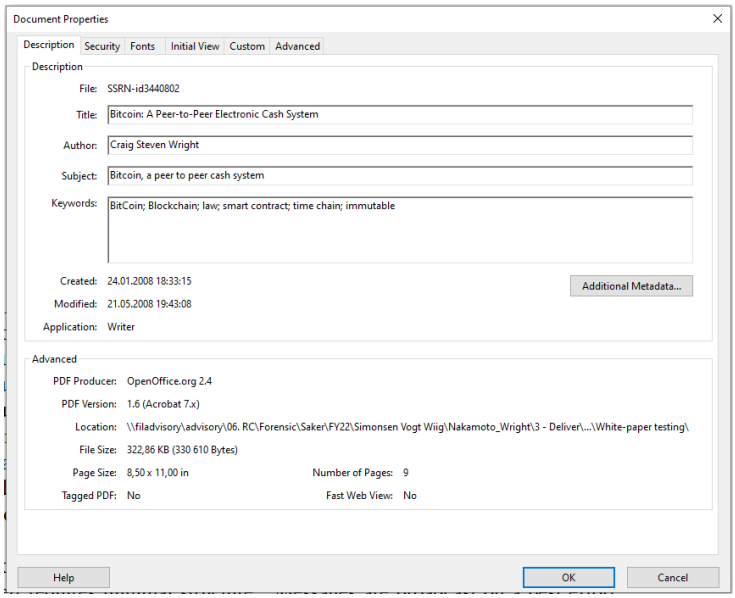


4.3.4 FTK metadata

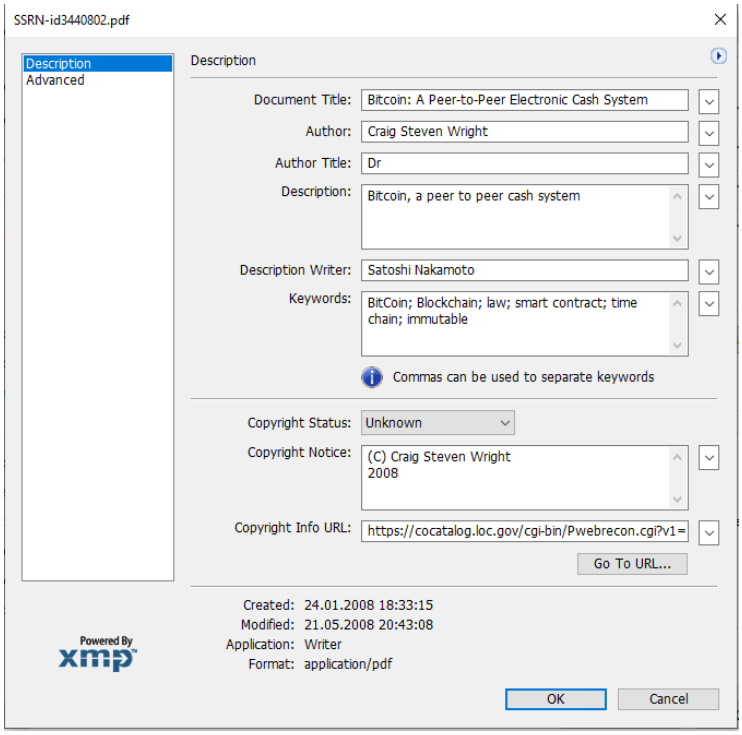


4.4 Vedlegg 4: Metadata «SSRN-id3440802.pdf»

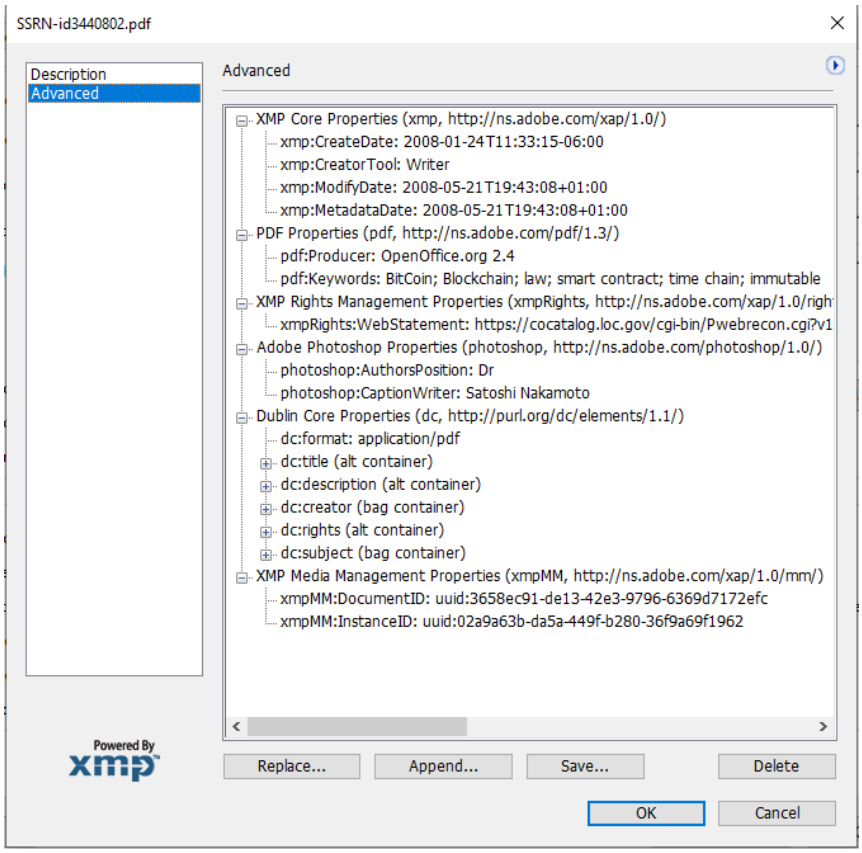
4.4.1 Adobe metadata



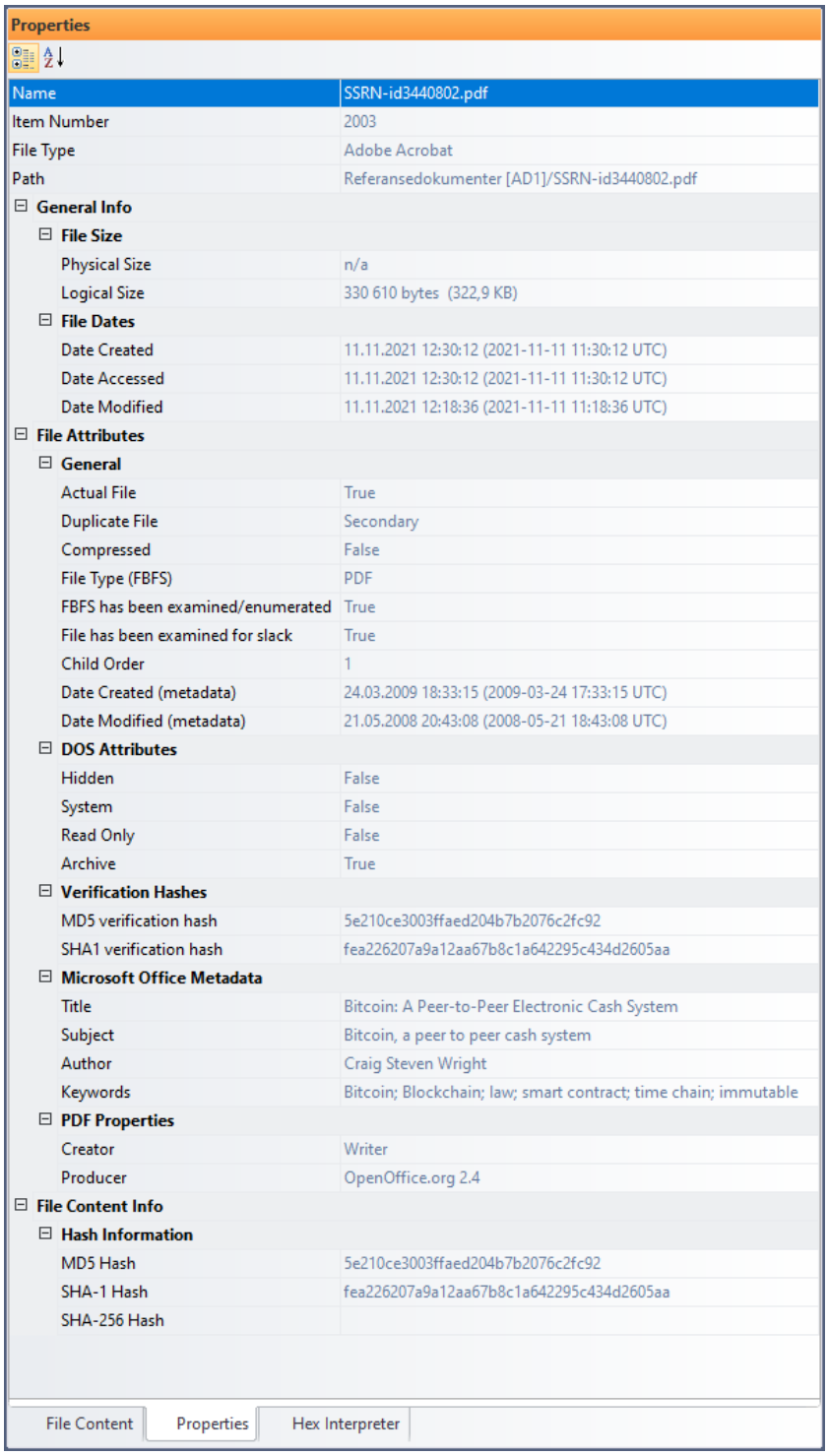
4.4.2 XMP description



4.4.3 XMP Advanced

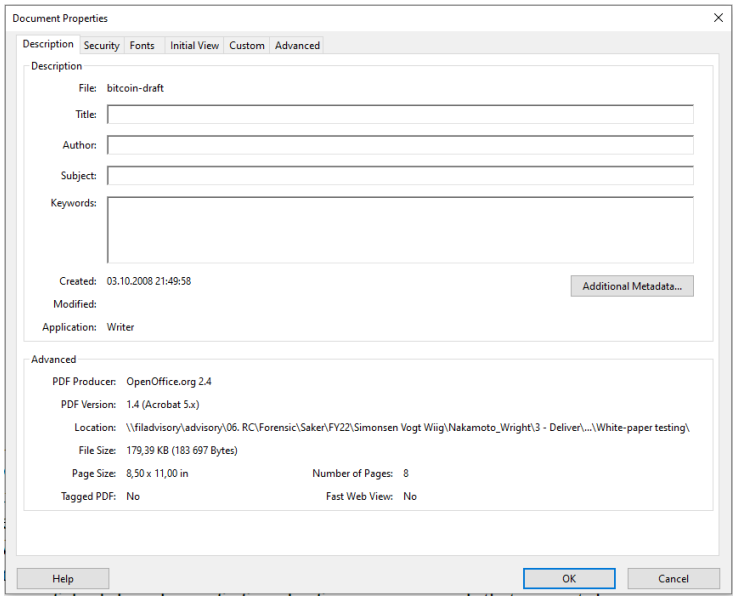


4.4.4 FTK metadata

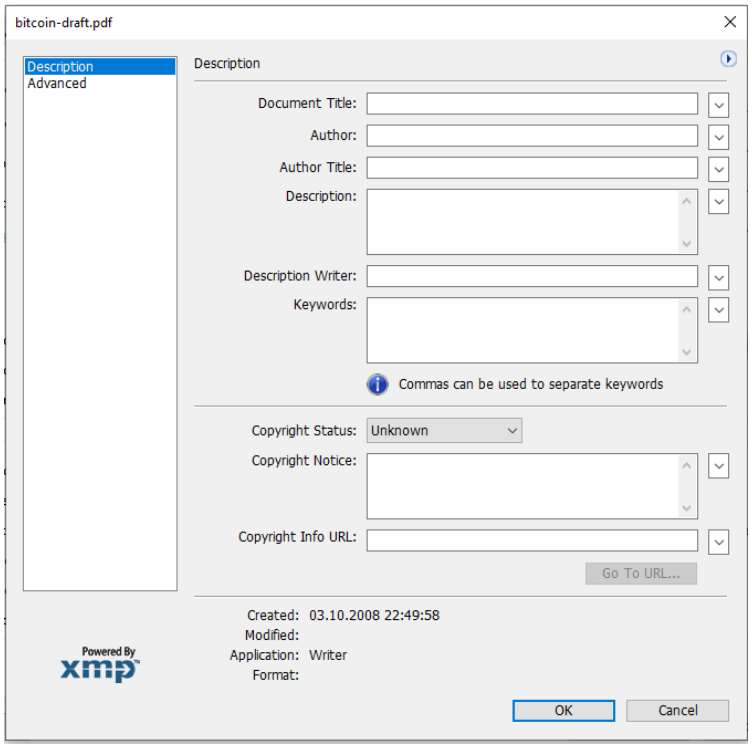


4.5 Vedlegg 5: Metadata «bitcoin-draft.pdf»

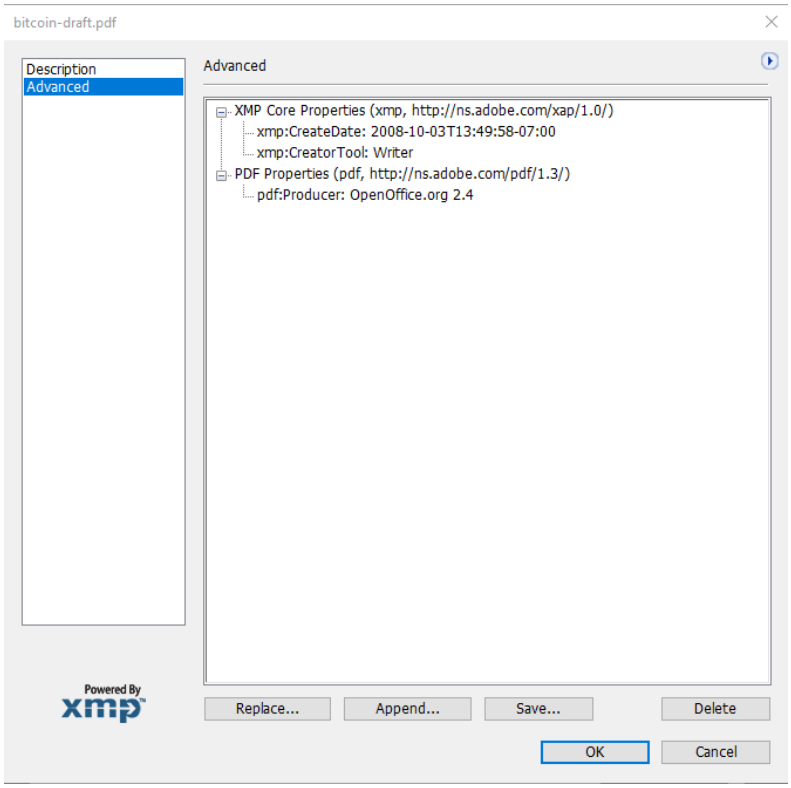
4.5.1 Adobe metadata



4.5.2 XMP description

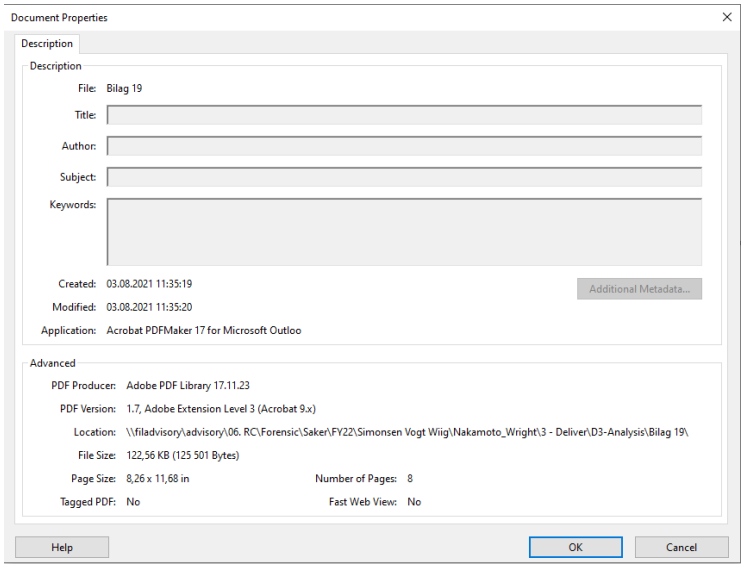


4.5.3 XMP Advanced

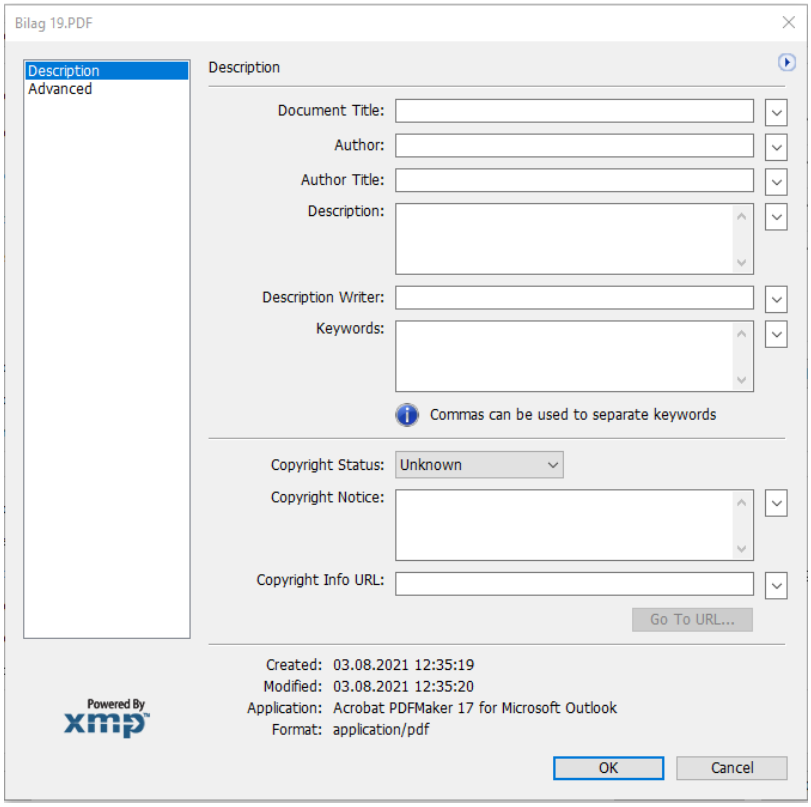


4.6 Vedlegg 6: Metadata «Bilag 19.pdf»

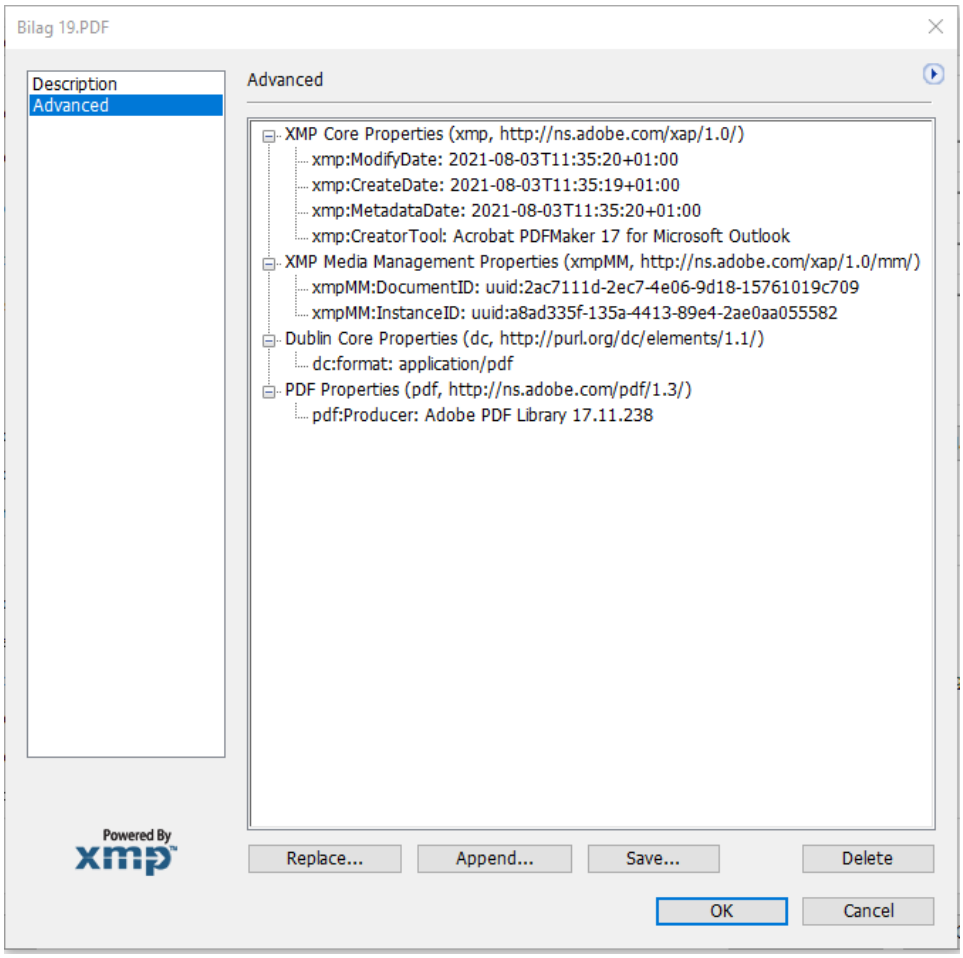
4.6.1 Adobe metadata



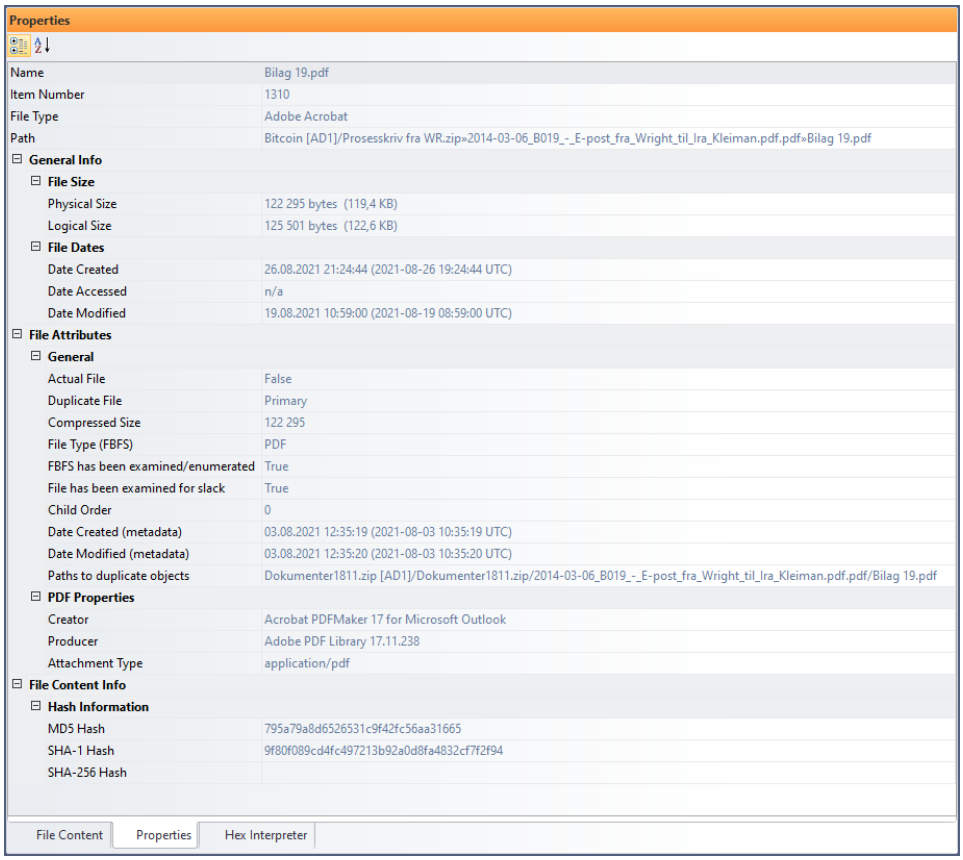
4.6.2 XMP Description



4.6.3 XMP Advanced

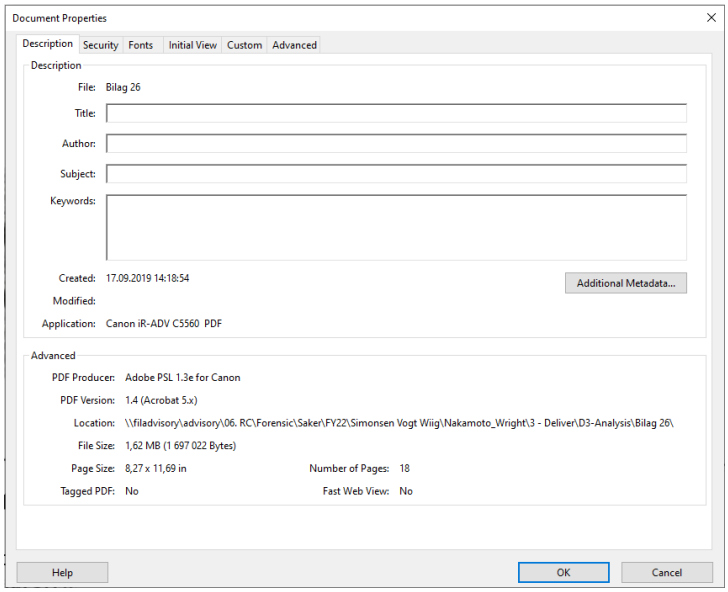


4.6.4 FTK metadata

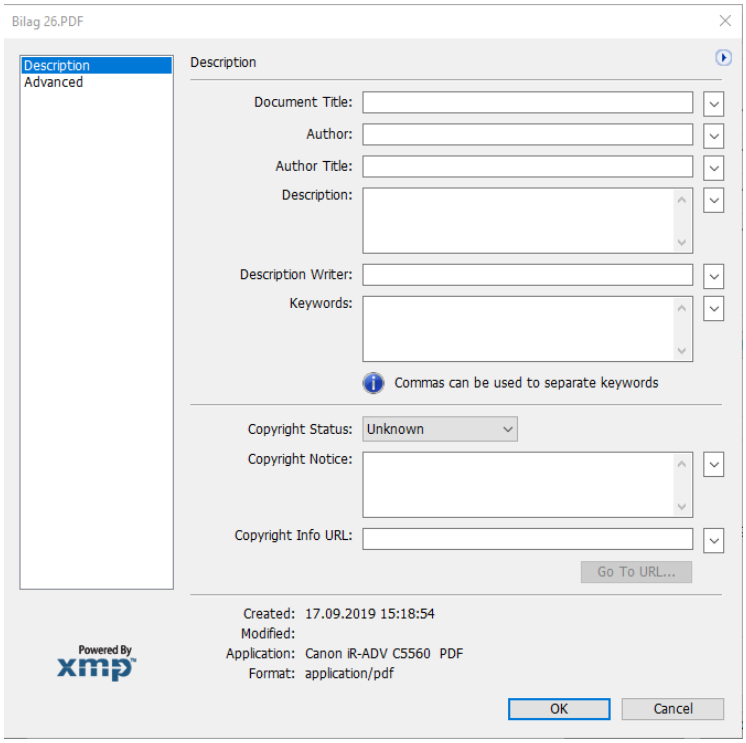


4.7 Vedlegg 7: Metadata «Bilag 26.pdf»

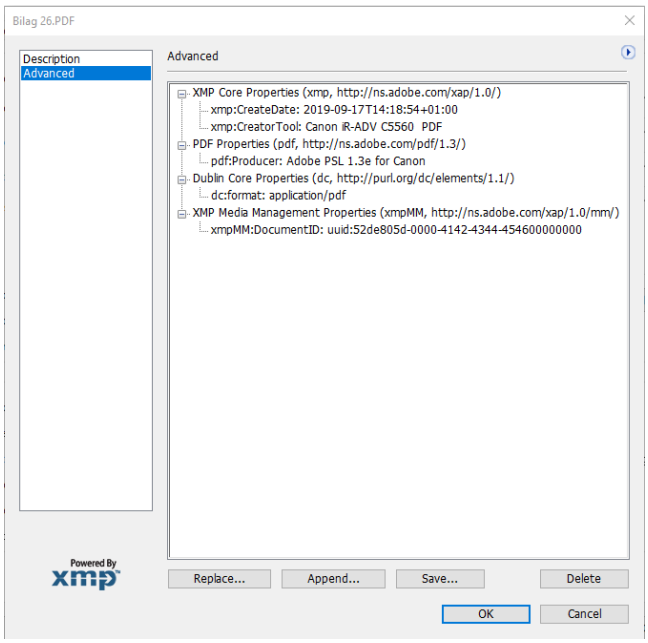
4.7.1 Adobe metadata



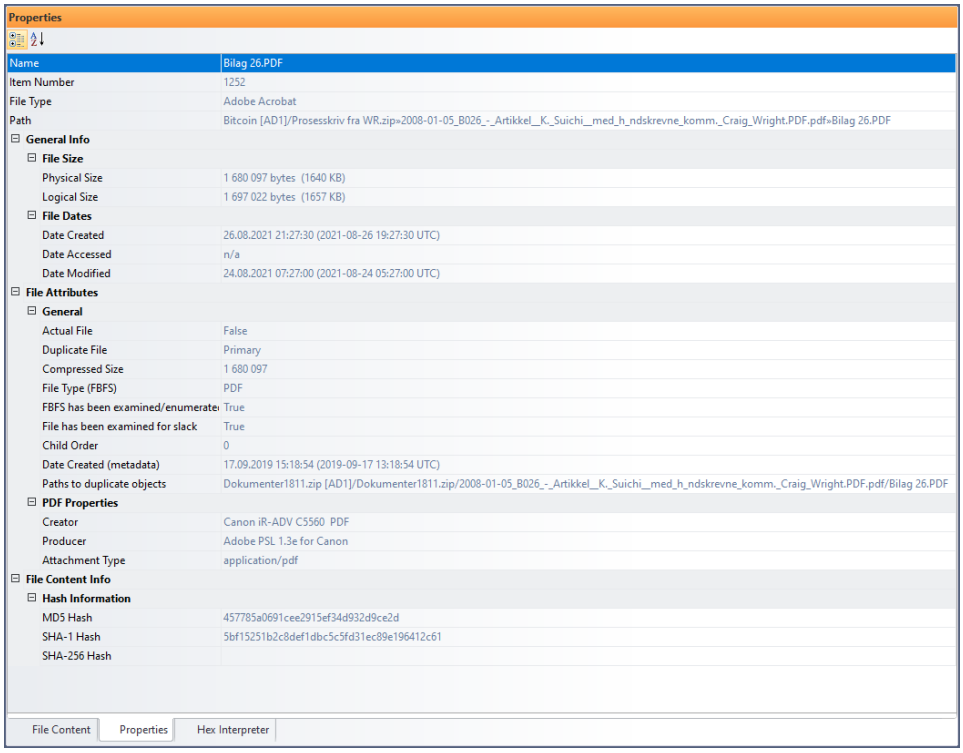
4.7.2 XMP Description



4.7.3 XMP Advanced

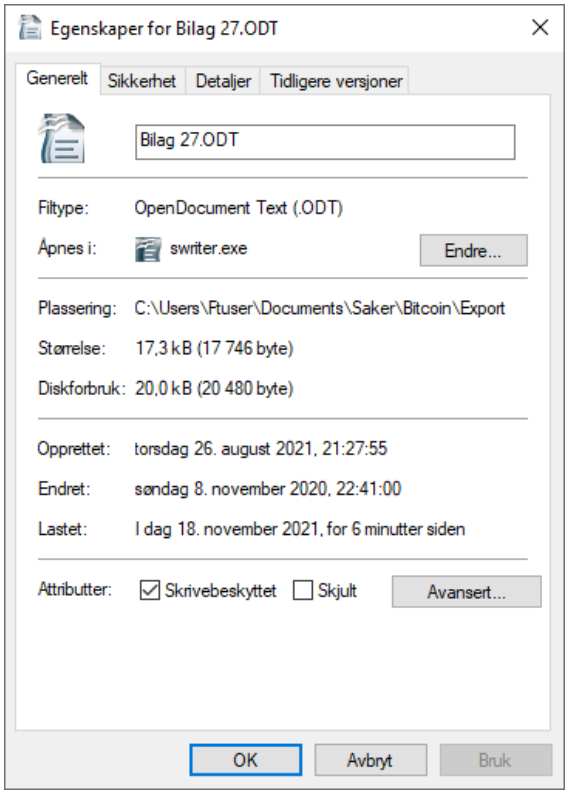


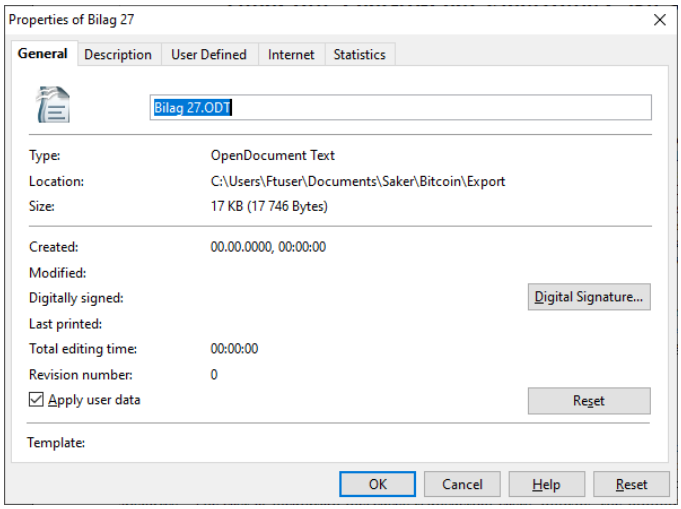
4.7.4 FTK metadata

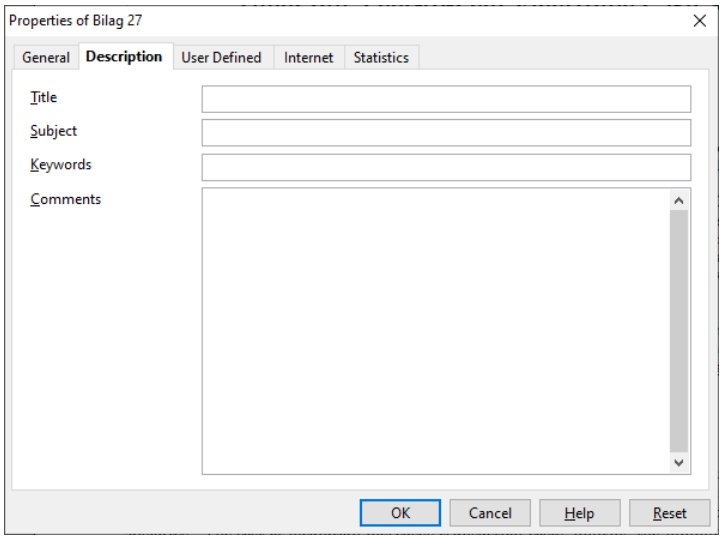


4.8 Vedlegg 8: Metadata «Bilag 27.odt»

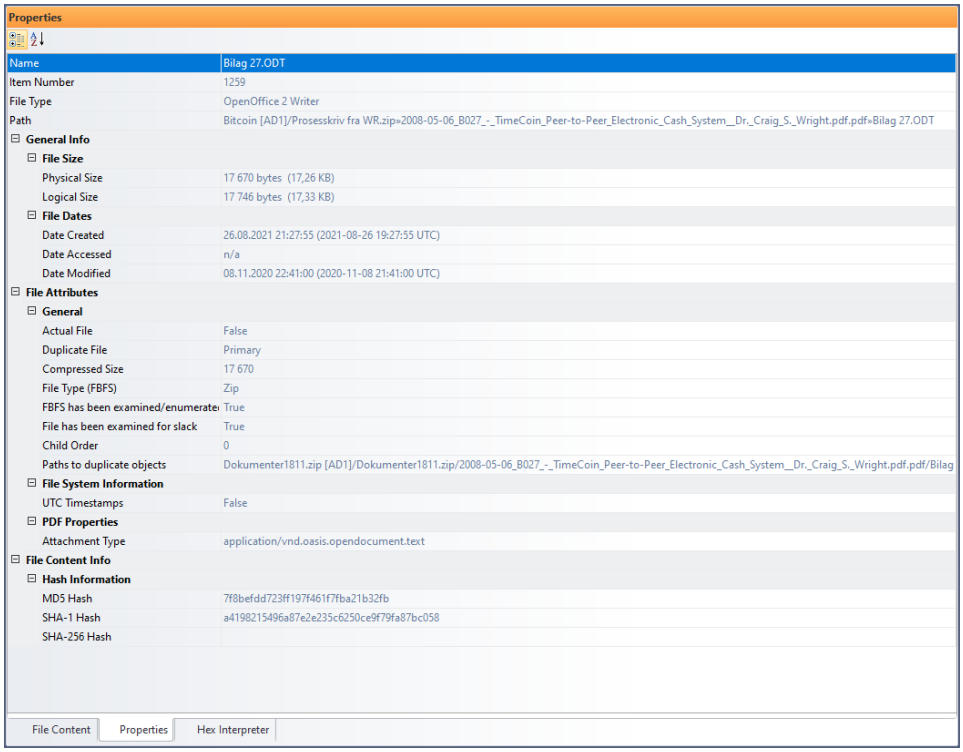
4.8.1 Open Office egenskaper





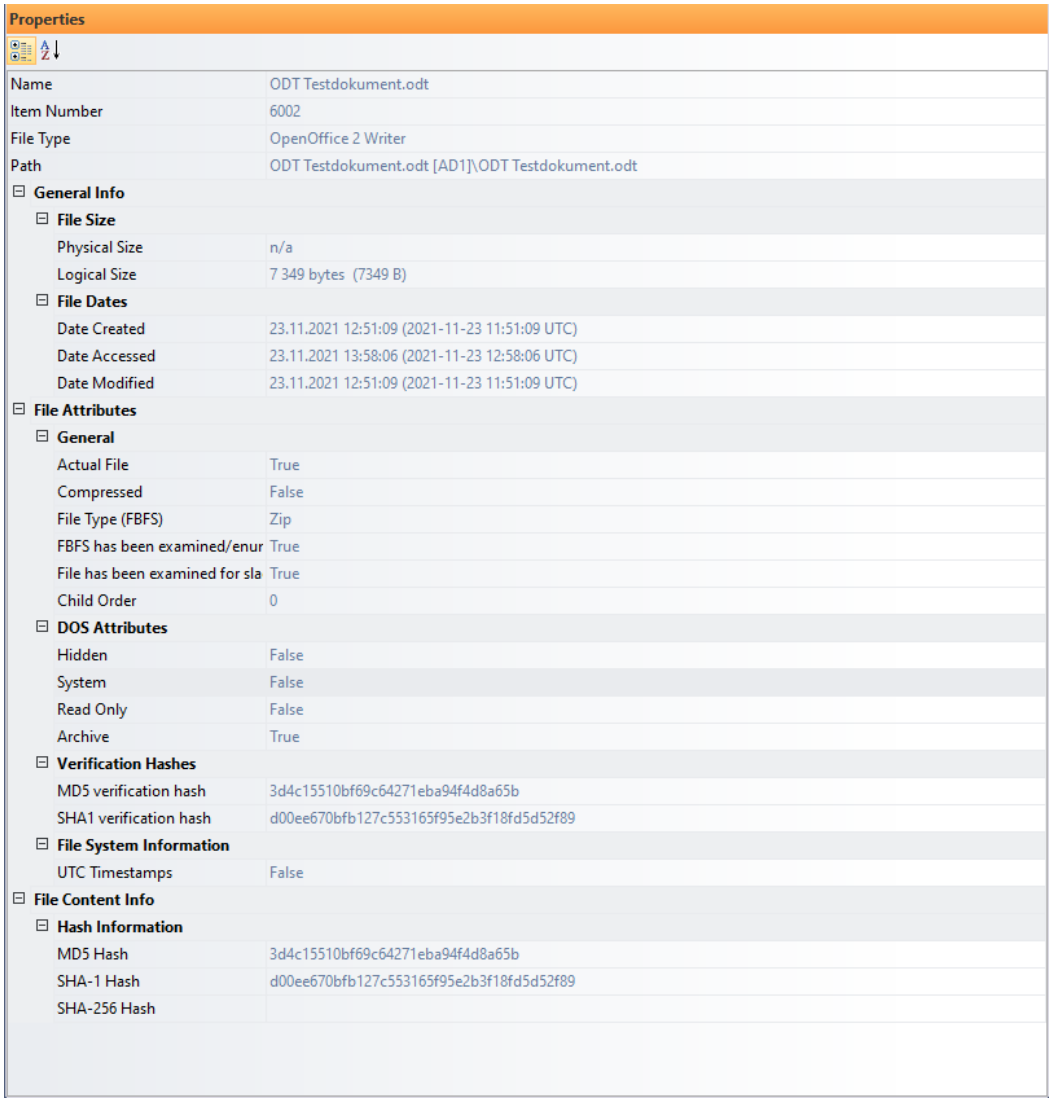


4.8.2 FTK metadata



4.9 Vedlegg 9: Metadata «ODT Testdokument.odt»

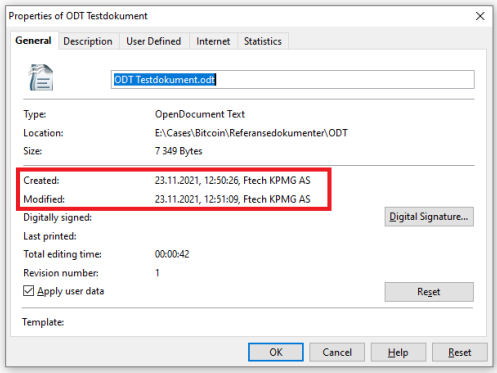
4.9.1 FTK metadata



Properties	
Name	ODT Testdokument.odt
Item Number	6002
File Type	OpenOffice 2 Writer
Path	ODT Testdokument.odt [AD1]\ODT Testdokument.odt
General Info	
File Size	
Physical Size	n/a
Logical Size	7 349 bytes (7349 B)
File Dates	
Date Created	23.11.2021 12:51:09 (2021-11-23 11:51:09 UTC)
Date Accessed	23.11.2021 13:58:06 (2021-11-23 12:58:06 UTC)
Date Modified	23.11.2021 12:51:09 (2021-11-23 11:51:09 UTC)
File Attributes	
General	
Actual File	True
Compressed	False
File Type (FBFS)	Zip
FBFS has been examined/enur	True
File has been examined for sla	True
Child Order	0
DOS Attributes	
Hidden	False
System	False
Read Only	False
Archive	True
Verification Hashes	
MD5 verification hash	3d4c15510bf69c64271eba94f4d8a65b
SHA1 verification hash	d00ee670bfb127c553165f95e2b3f18fd5d52f89
File System Information	
UTC Timestamps	False
File Content Info	
Hash Information	
MD5 Hash	3d4c15510bf69c64271eba94f4d8a65b
SHA-1 Hash	d00ee670bfb127c553165f95e2b3f18fd5d52f89
SHA-256 Hash	

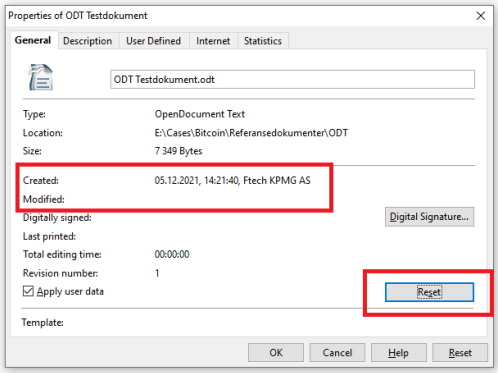
4.9.2 Datoer fra Open Office

Testdokument



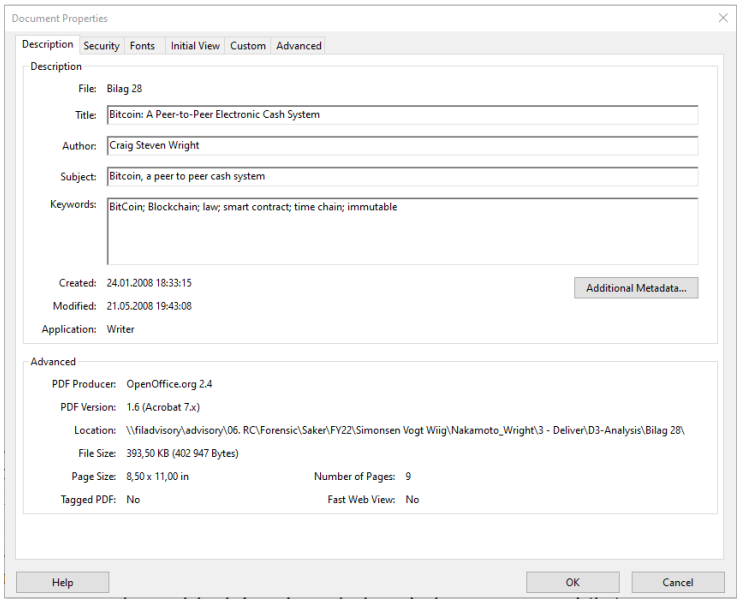
4.9.3 Datoer fra Open Office etter reset

Testdokument

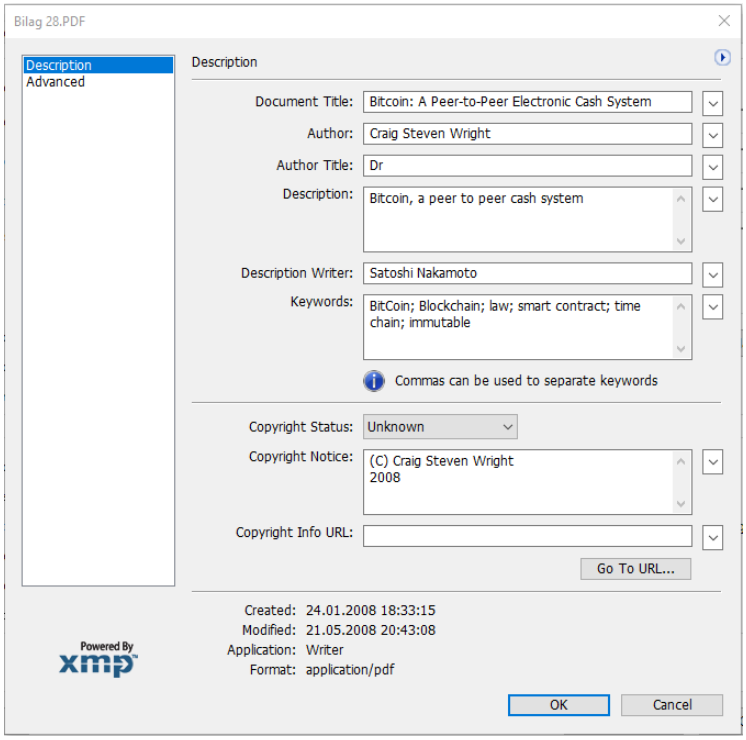


4.10 Vedlegg 10: Metadata «Bilag 28.pdf»

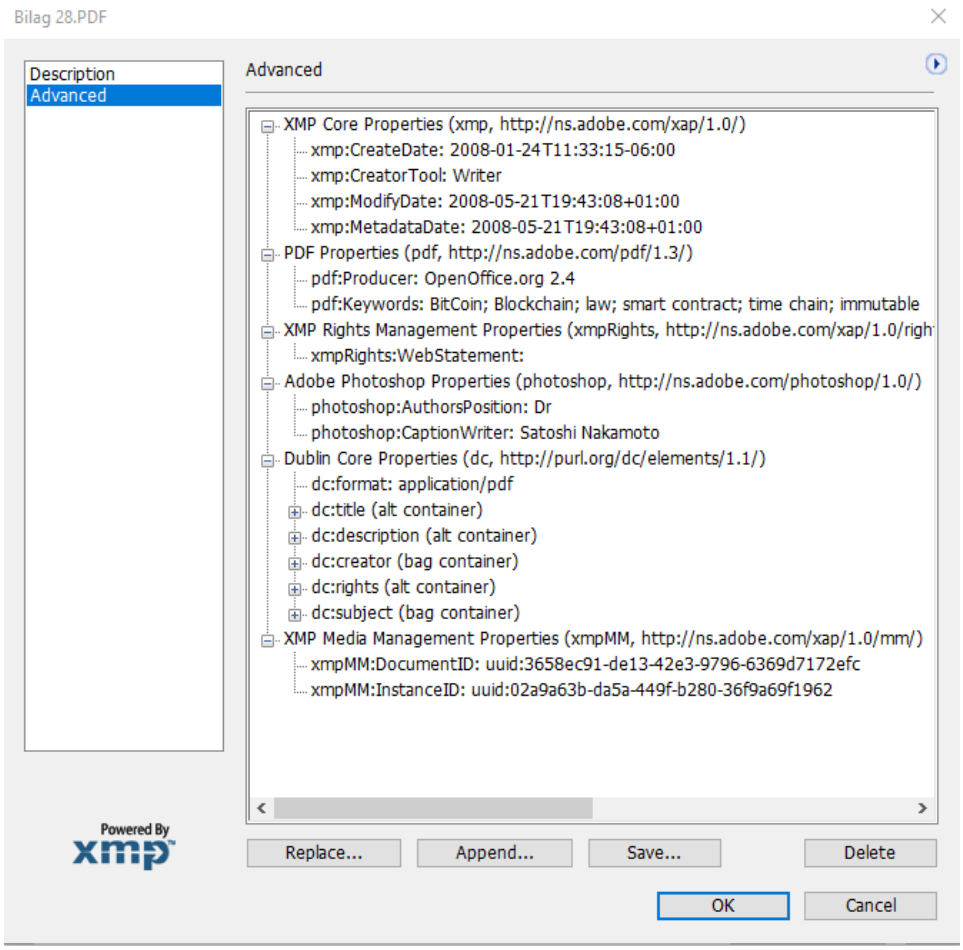
4.10.1 Adobe metadata



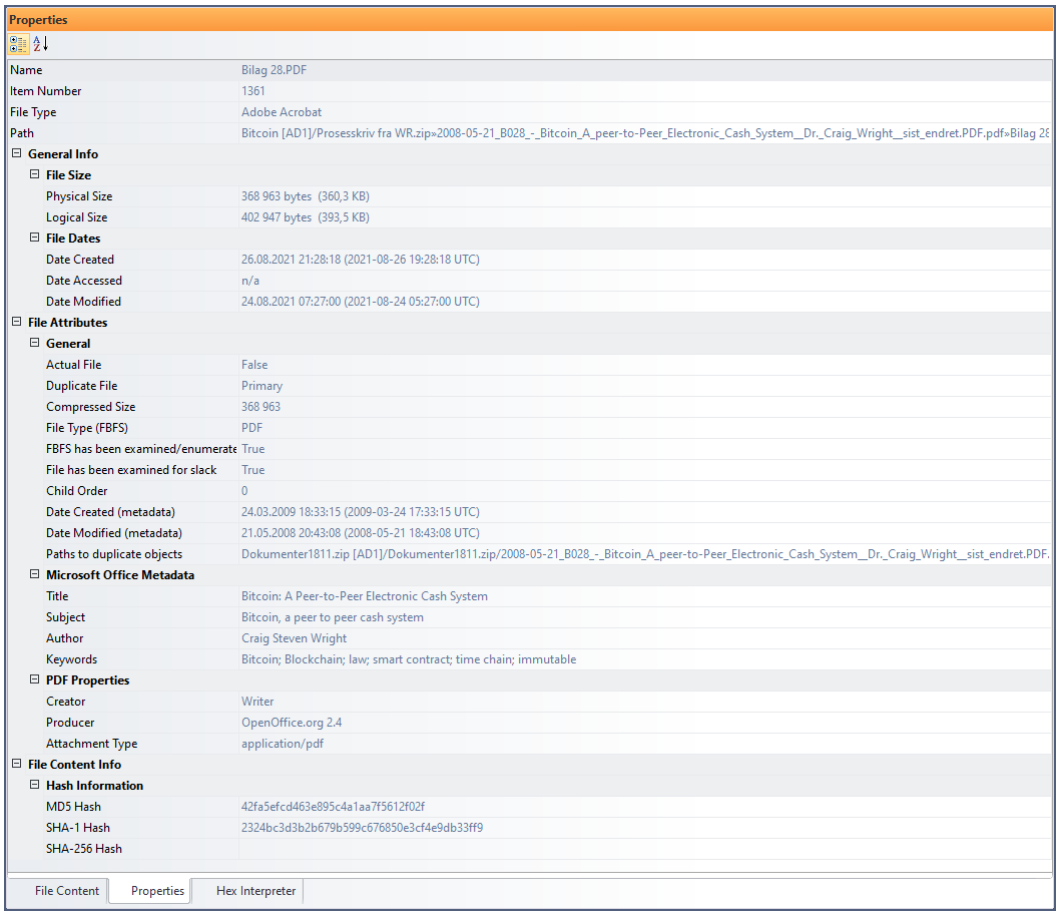
4.10.2 XMP description



4.10.3 XMP Advanced

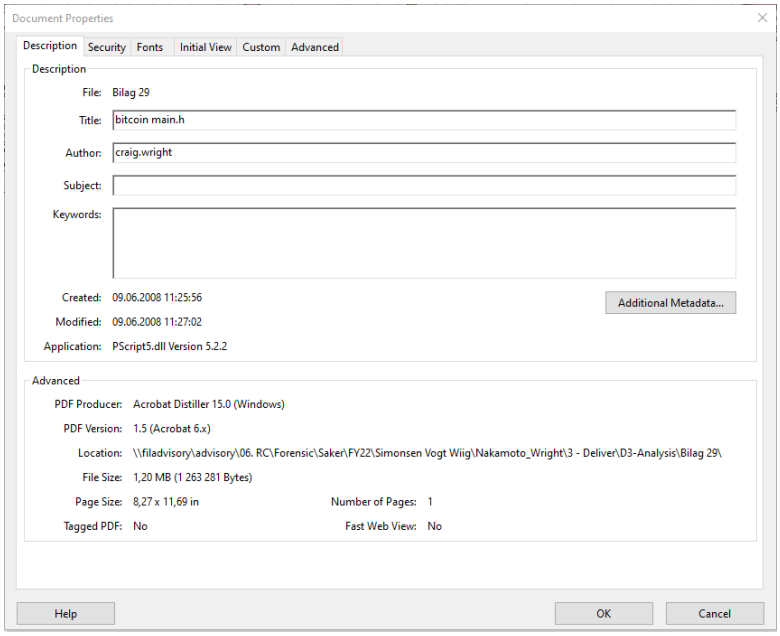


4.10.4 FTK metadata

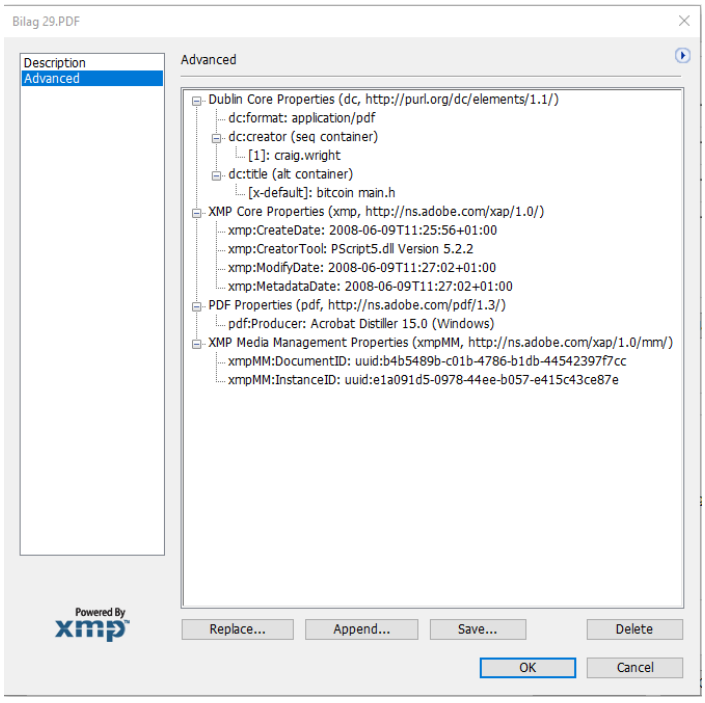


4.11 Vedlegg 11: Metadata «Bilag 29.pdf»

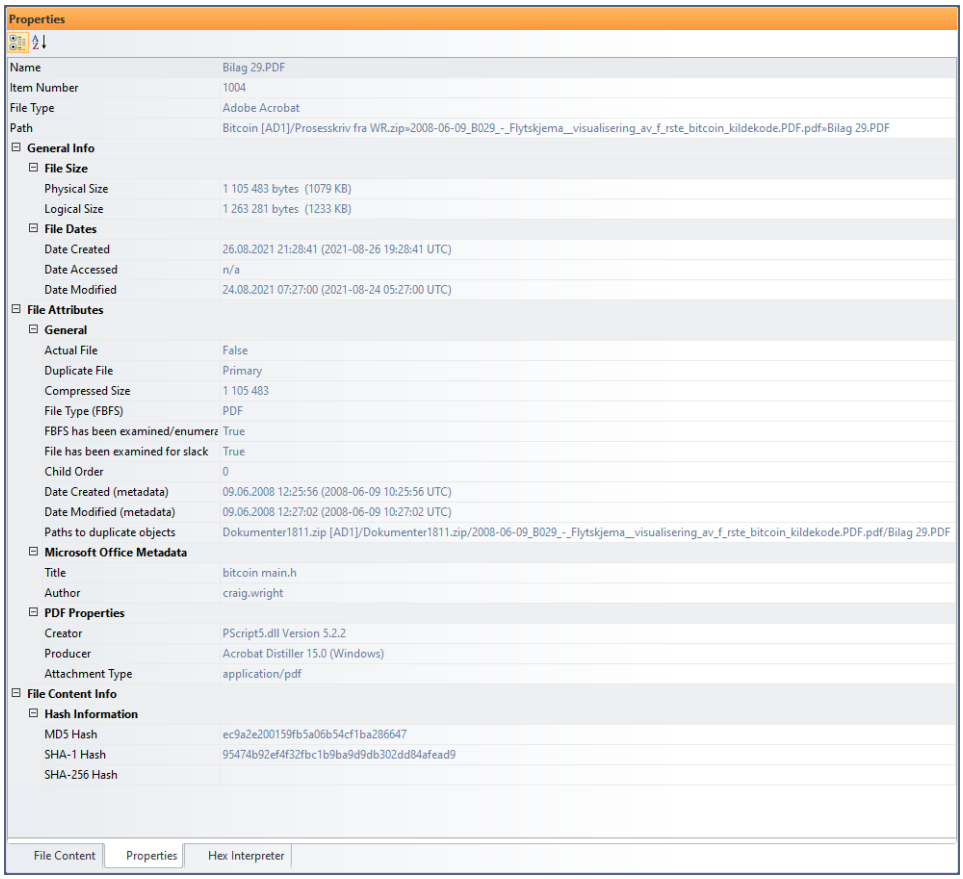
4.11.1 Adobe metadata



4.11.2 XMP Metadata

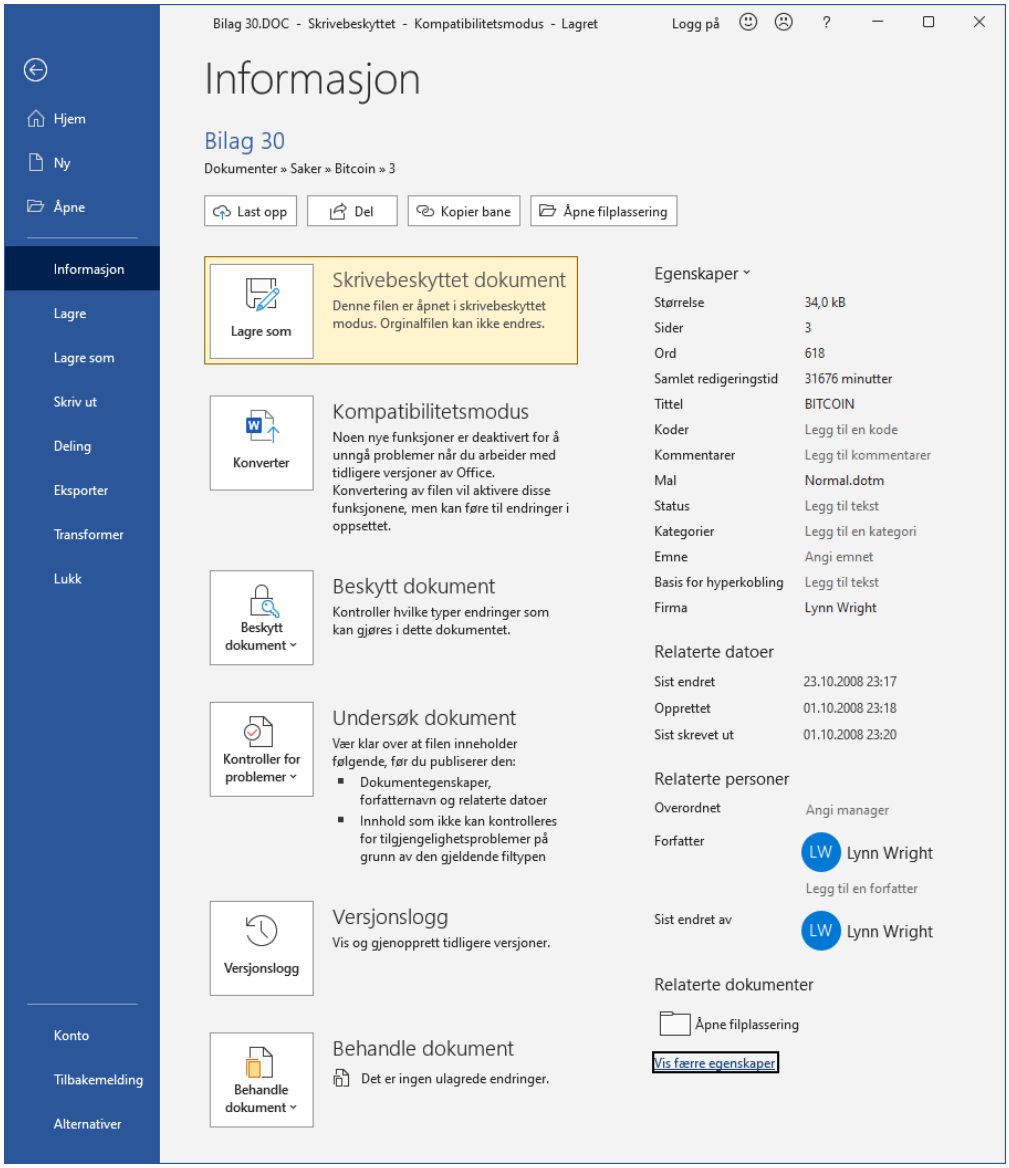


4.11.3 FTK metadata

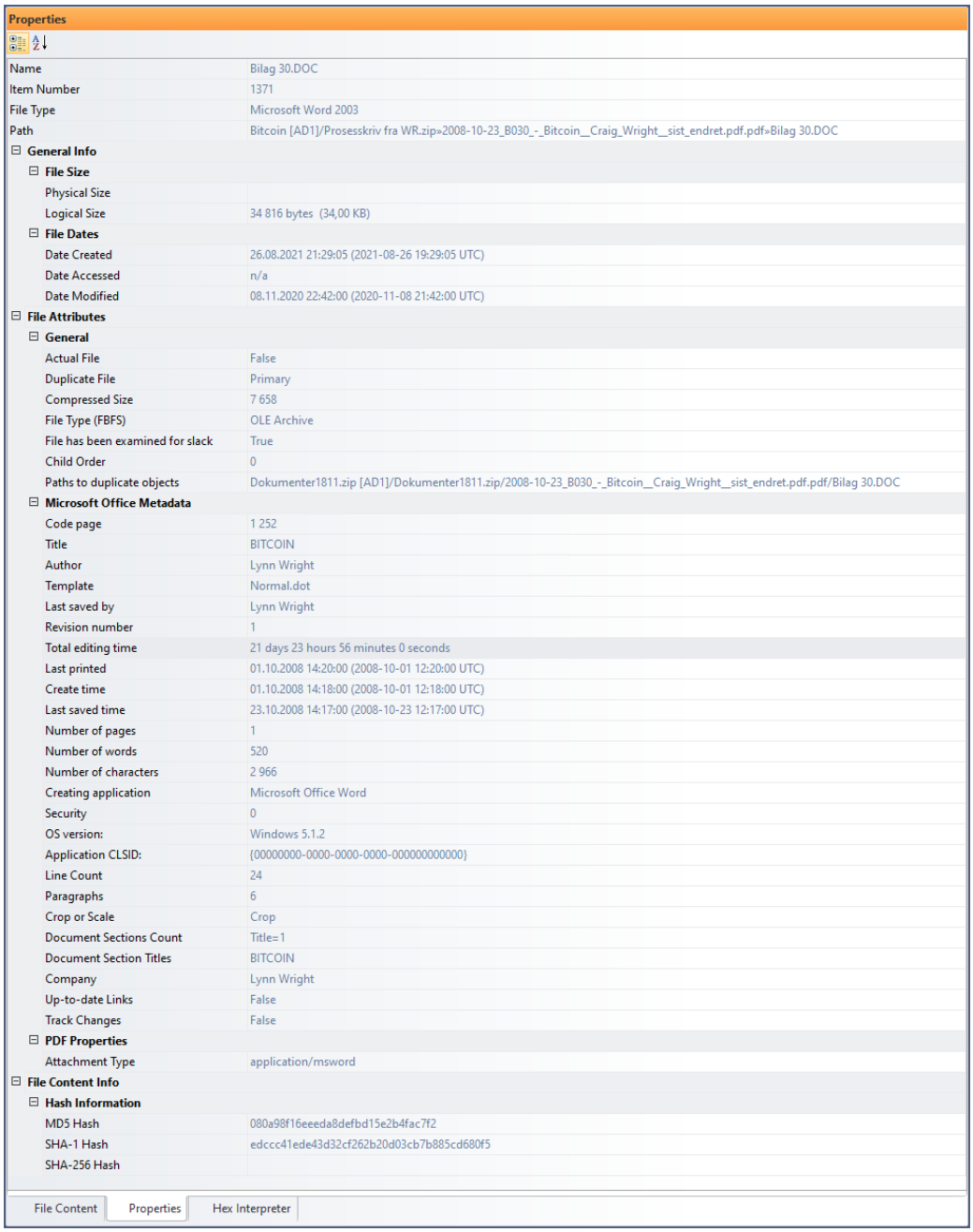


4.12 Vedlegg 12: Metadata «Bilag 30.doc»

4.12.1 MS Word informasjon



4.12.2 FTK metadata



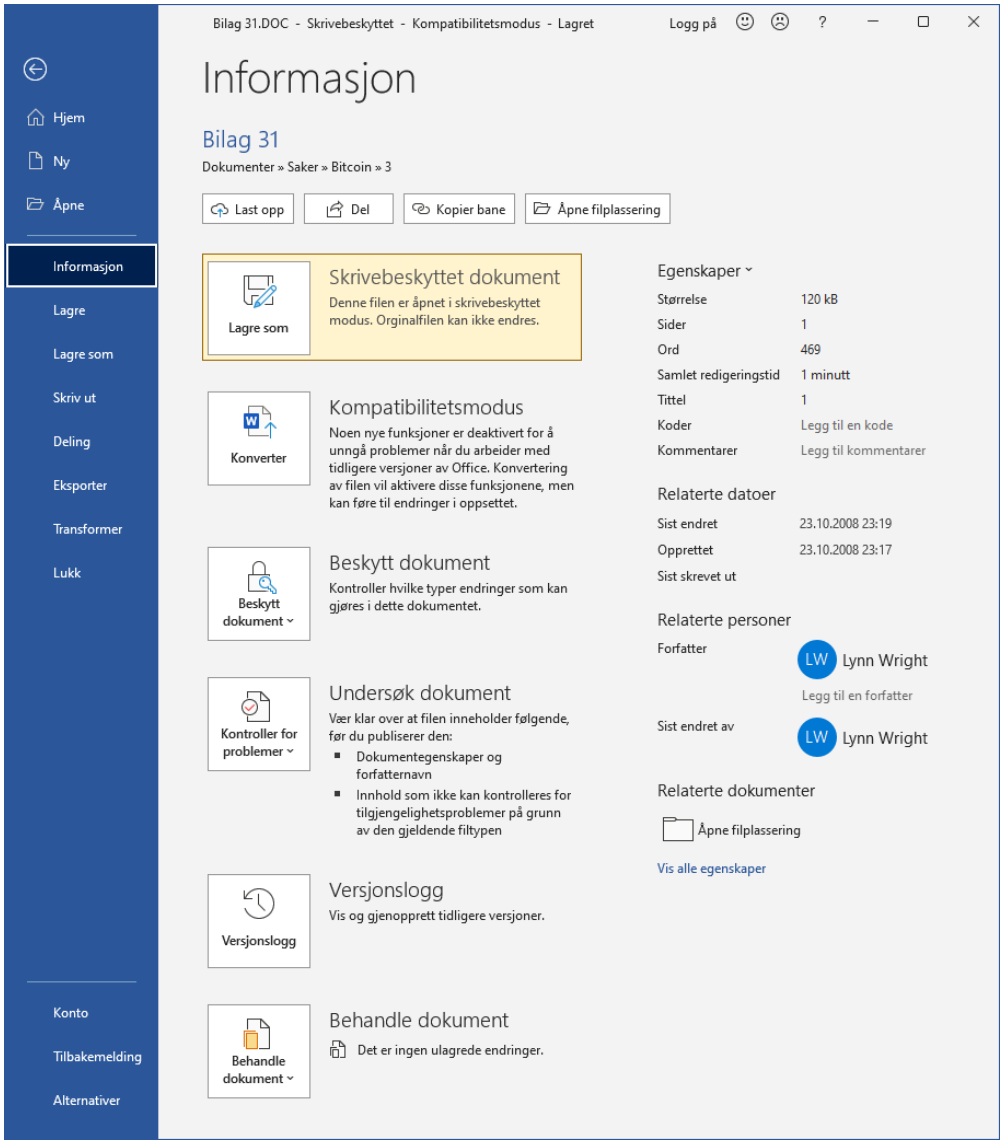
Properties

Name	Bilag 30.DOC
Item Number	1371
File Type	Microsoft Word 2003
Path	Bitcoin [AD1]/Prosesskriv fra WR.zip»2008-10-23_B030_-_Bitcoin__Craig_Wright__sist_endret.pdf.pdf»Bilag 30.DOC
General Info	
File Size	
Physical Size	
Logical Size	34 816 bytes (34,00 KB)
File Dates	
Date Created	26.08.2021 21:29:05 (2021-08-26 19:29:05 UTC)
Date Accessed	n/a
Date Modified	08.11.2020 22:42:00 (2020-11-08 21:42:00 UTC)
File Attributes	
General	
Actual File	False
Duplicate File	Primary
Compressed Size	7 658
File Type (FBFS)	OLE Archive
File has been examined for slack	True
Child Order	0
Paths to duplicate objects	Dokumenter1811.zip [AD1]/Dokumenter1811.zip/2008-10-23_B030_-_Bitcoin__Craig_Wright__sist_endret.pdf.pdf/Bilag 30.DOC
Microsoft Office Metadata	
Code page	1 252
Title	BITCOIN
Author	Lynn Wright
Template	Normal.dot
Last saved by	Lynn Wright
Revision number	1
Total editing time	21 days 23 hours 56 minutes 0 seconds
Last printed	01.10.2008 14:20:00 (2008-10-01 12:20:00 UTC)
Create time	01.10.2008 14:18:00 (2008-10-01 12:18:00 UTC)
Last saved time	23.10.2008 14:17:00 (2008-10-23 12:17:00 UTC)
Number of pages	1
Number of words	520
Number of characters	2 966
Creating application	Microsoft Office Word
Security	0
OS version:	Windows 5.1.2
Application CLSID:	{00000000-0000-0000-0000-000000000000}
Line Count	24
Paragraphs	6
Crop or Scale	Crop
Document Sections Count	Title=1
Document Section Titles	BITCOIN
Company	Lynn Wright
Up-to-date Links	False
Track Changes	False
PDF Properties	
Attachment Type	application/msword
File Content Info	
Hash Information	
MD5 Hash	080a98f16eeeda8defbd15e2b4fac7f2
SHA-1 Hash	edccc41ede43d32cf262b20d03cb7b885cd680f5
SHA-256 Hash	

File Content | Properties | Hex Interpreter

4.13 Vedlegg 13: Metadata «Bilag 31.doc»

4.13.1 MS Word informasjon



4.13.2 FTK metadata

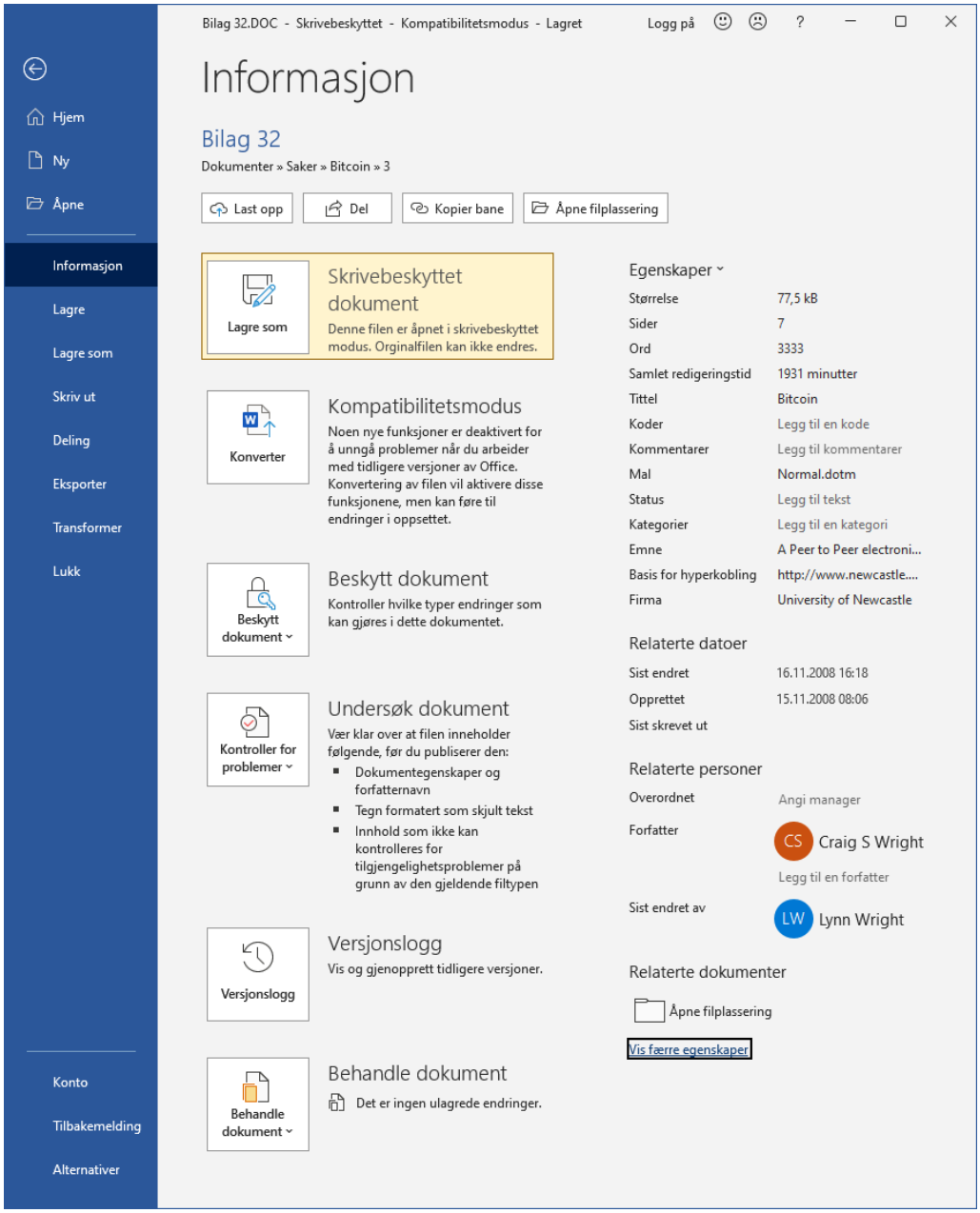
Properties

Name	Bilag 31.DOC
Item Number	1254
File Type	Microsoft Word 2003
Path	Bitcoin [AD1]/Prosesskriv fra WR.zip»2008-10-23_B031_-_Bitcoin__law___Craig_Wright__sist_endret.pdf.pdf»Bilag 31.DOC
General Info	
File Size	
Physical Size	
Logical Size	122 880 bytes (120,0 KB)
File Dates	
Date Created	26.08.2021 21:29:39 (2021-08-26 19:29:39 UTC)
Date Accessed	n/a
Date Modified	08.11.2020 22:42:00 (2020-11-08 21:42:00 UTC)
File Attributes	
General	
Actual File	False
Duplicate File	Primary
Compressed Size	11 908
File Type (FBFS)	OLE Archive
File has been examined for slack	True
Child Order	0
Paths to duplicate objects	Dokumenter1811.zip [AD1]/Dokumenter1811.zip/2008-10-23_B031_-_Bitcoin__law___Craig_Wright__sist_endret.pdf.pdf/Bilag 31.DOC
Microsoft Office Metadata	
Code page	1 252
Title	1
Author	Lynn Wright
Template	Normal.dot
Last saved by	Lynn Wright
Revision number	1
Total editing time	1 minutes 0 seconds
Create time	23.10.2008 14:17:00 (2008-10-23 12:17:00 UTC)
Last saved time	23.10.2008 14:19:00 (2008-10-23 12:19:00 UTC)
Number of pages	1
Number of words	434
Number of characters	2 479
Creating application	Microsoft Office Word
Security	0
OS version:	Windows 5.1.2
Application CLSID:	{00000000-0000-0000-0000-000000000000}
Line Count	20
Paragraphs	5
Crop or Scale	Crop
Document Sections Count	Title=1
Document Section Titles	1
Company	Lynn Wright
Up-to-date Links	False
Track Changes	False
PDF Properties	
Attachment Type	application/msword
File Content Info	
Hash Information	
MD5 Hash	4128db5c270060e1464b9d516b3de8ea
SHA-1 Hash	abb31e6493ba6031f85b1ab4e2420147edc97b3d
SHA-256 Hash	

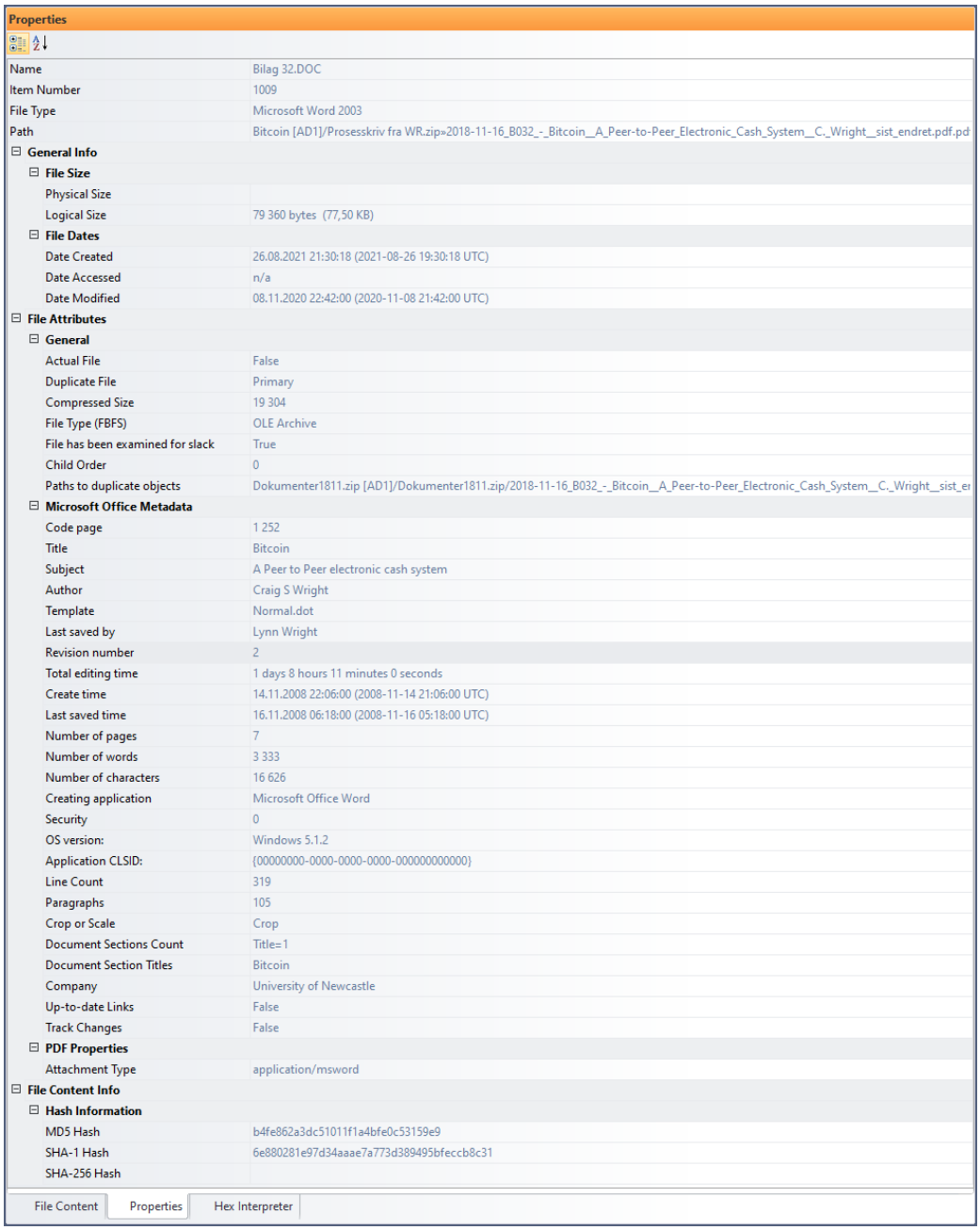
File Content | Properties | Hex Interpreter

4.14 Vedlegg 14: Metadata «Bilag 32.doc»

4.14.1 MS Word informasjon



4.14.2 FTK metadata



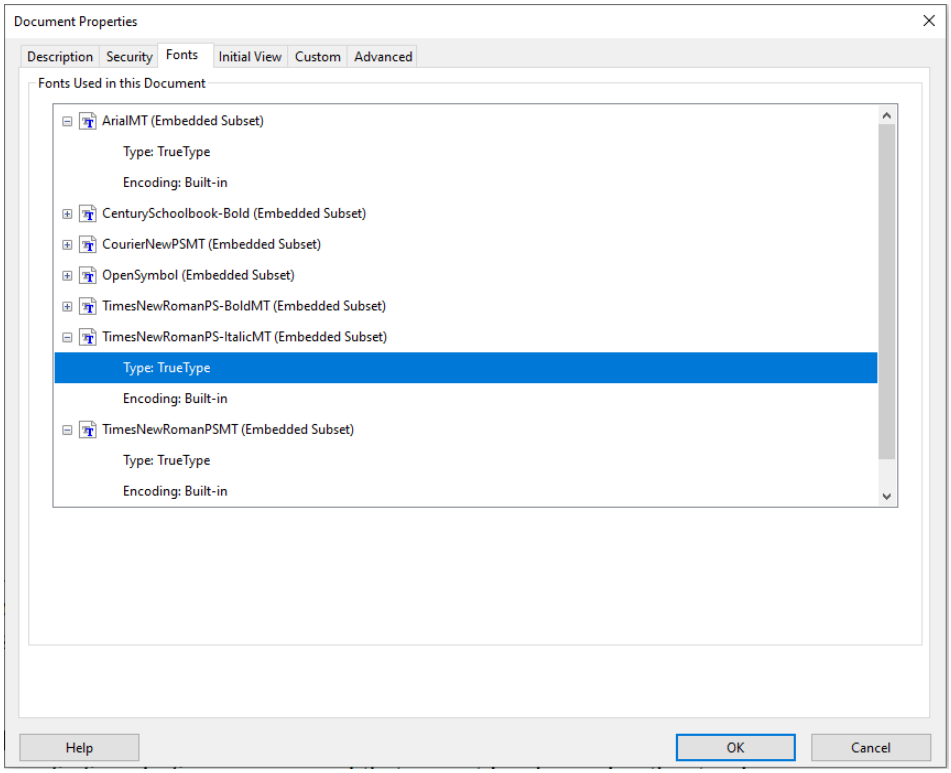
Properties

Name	Bilag 32.DOC
Item Number	1009
File Type	Microsoft Word 2003
Path	Bitcoin [AD1]/Prosesskriv fra WR.zip»2018-11-16_B032_-_Bitcoin__A_Peer-to-Peer_Electronic_Cash_System__C._Wright__sist_endret.pdf.pd
General Info	
File Size	
Physical Size	
Logical Size	79 360 bytes (77,50 KB)
File Dates	
Date Created	26.08.2021 21:30:18 (2021-08-26 19:30:18 UTC)
Date Accessed	n/a
Date Modified	08.11.2020 22:42:00 (2020-11-08 21:42:00 UTC)
File Attributes	
General	
Actual File	False
Duplicate File	Primary
Compressed Size	19 304
File Type (FBFS)	OLE Archive
File has been examined for slack	True
Child Order	0
Paths to duplicate objects	Dokumenter1811.zip [AD1]/Dokumenter1811.zip/2018-11-16_B032_-_Bitcoin__A_Peer-to-Peer_Electronic_Cash_System__C._Wright__sist_er
Microsoft Office Metadata	
Code page	1 252
Title	Bitcoin
Subject	A Peer to Peer electronic cash system
Author	Craig S Wright
Template	Normal.dot
Last saved by	Lynn Wright
Revision number	2
Total editing time	1 days 8 hours 11 minutes 0 seconds
Create time	14.11.2008 22:06:00 (2008-11-14 21:06:00 UTC)
Last saved time	16.11.2008 06:18:00 (2008-11-16 05:18:00 UTC)
Number of pages	7
Number of words	3 333
Number of characters	16 626
Creating application	Microsoft Office Word
Security	0
OS version:	Windows 5.1.2
Application CLSID:	{00000000-0000-0000-0000-000000000000}
Line Count	319
Paragraphs	105
Crop or Scale	Crop
Document Sections Count	Title=1
Document Section Titles	Bitcoin
Company	University of Newcastle
Up-to-date Links	False
Track Changes	False
PDF Properties	
Attachment Type	application/msword
File Content Info	
Hash Information	
MD5 Hash	b4fe862a3dc51011f1a4bfe0c53159e9
SHA-1 Hash	6e880281e97d34aaae7a773d389495bfeccb8c31
SHA-256 Hash	

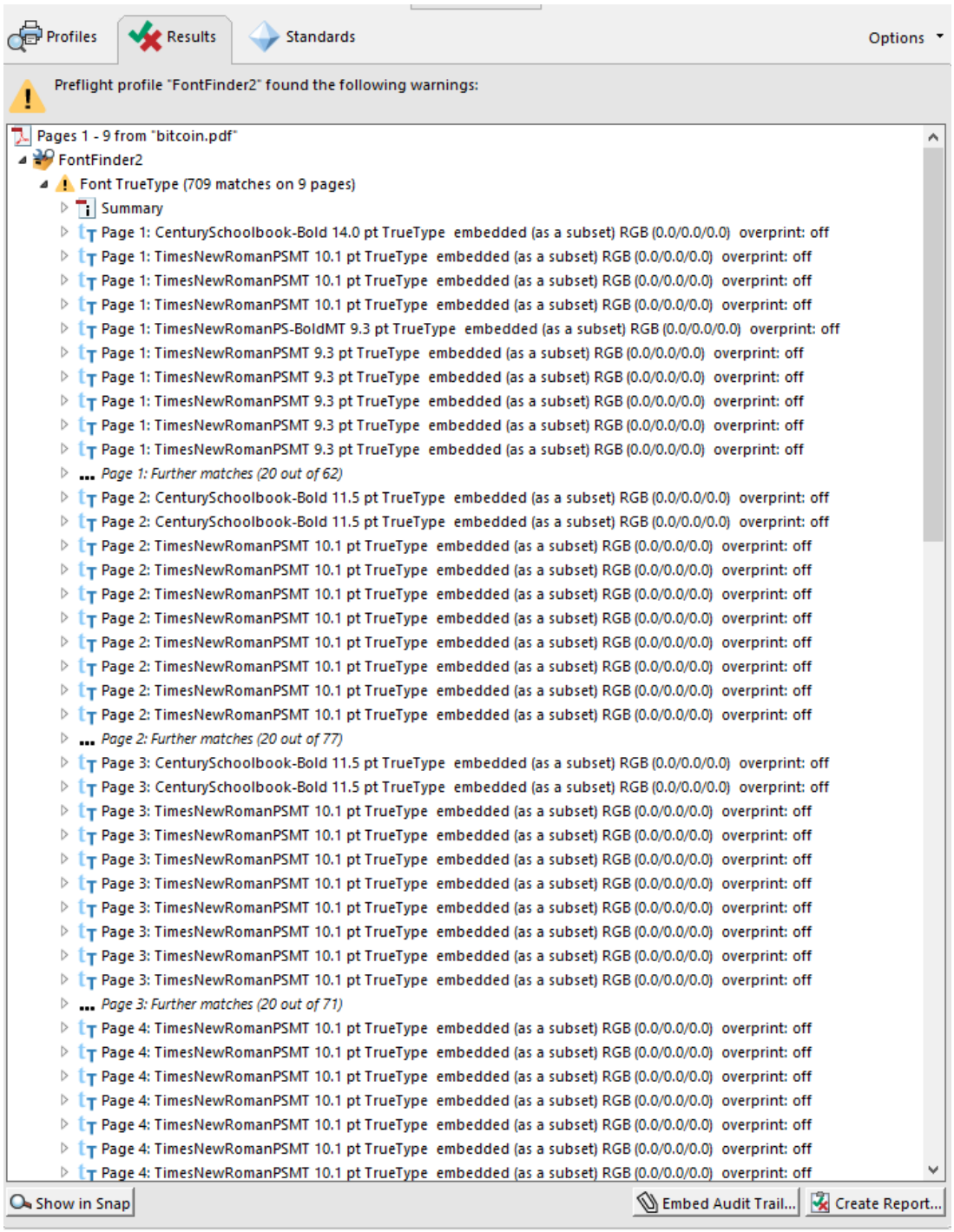
File Content | Properties | Hex Interpreter

4.15 Vedlegg 15: Innholdsanalyse «bitcoin.pdf»

4.15.1 Fontoversikt

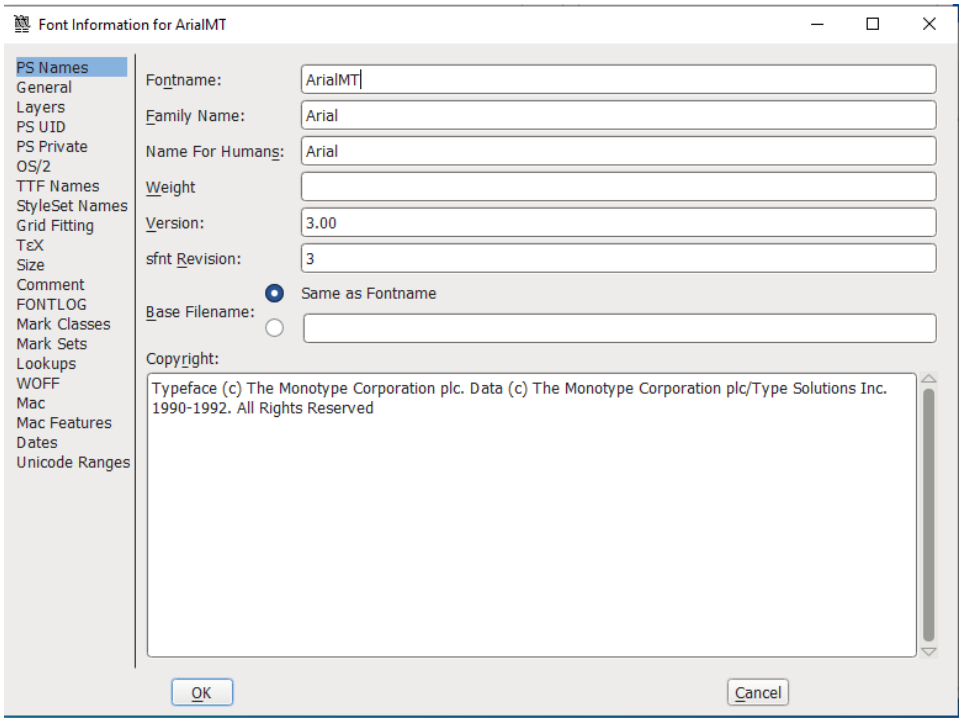


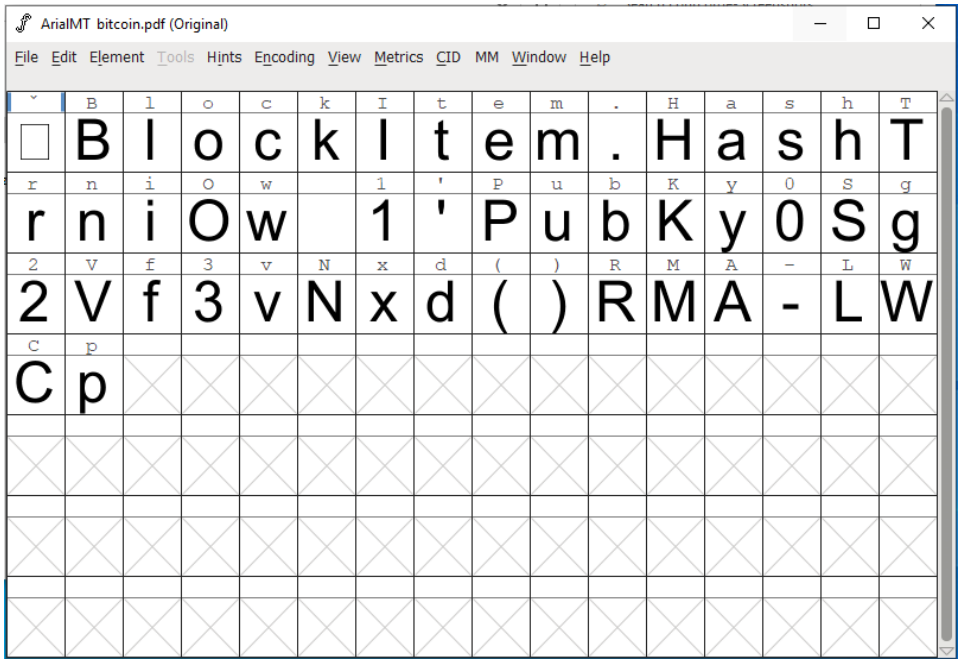
4.15.2 Font-typer i Adobe Preflight



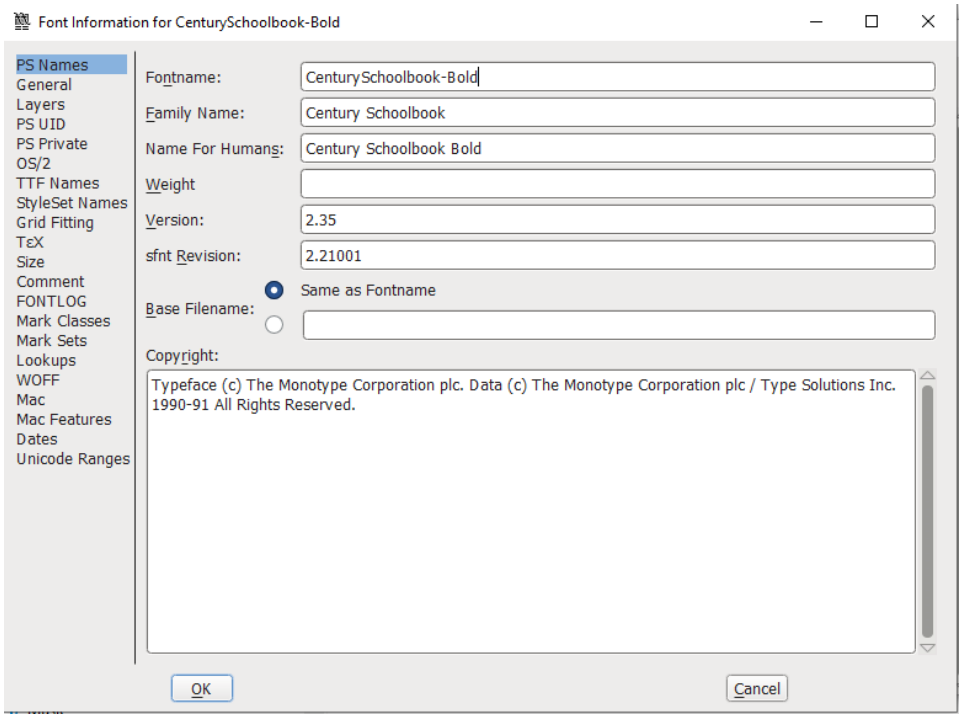
4.15.3 Font-metadata fra FontForge

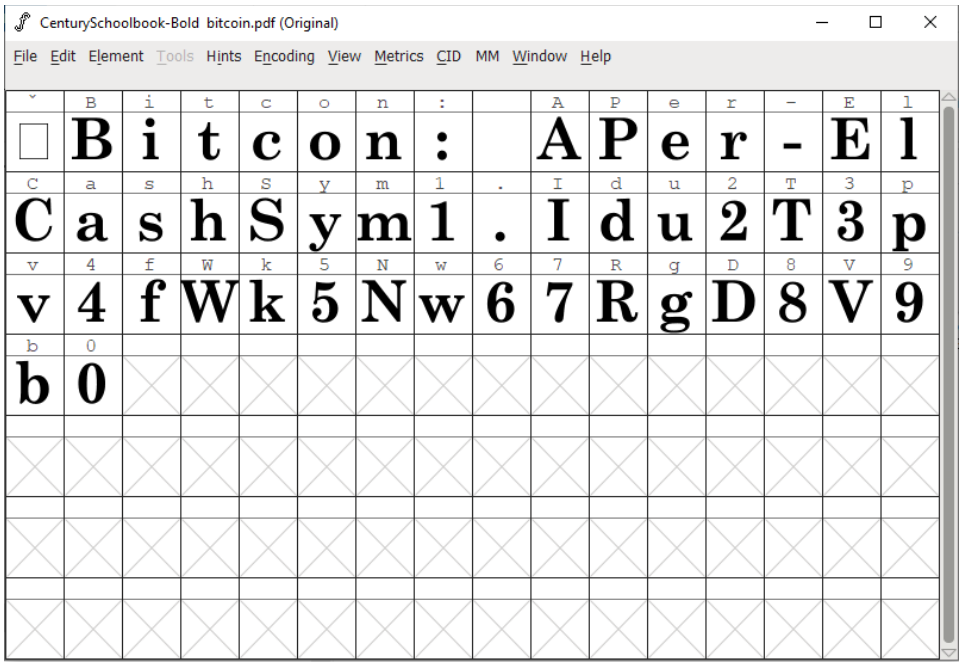
ArialMT



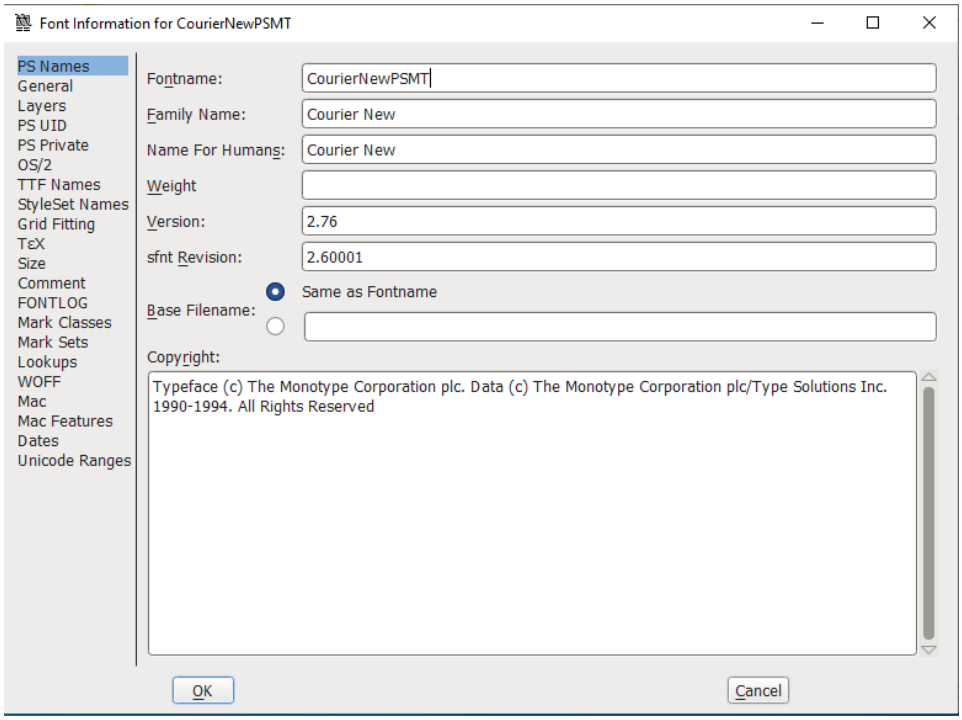


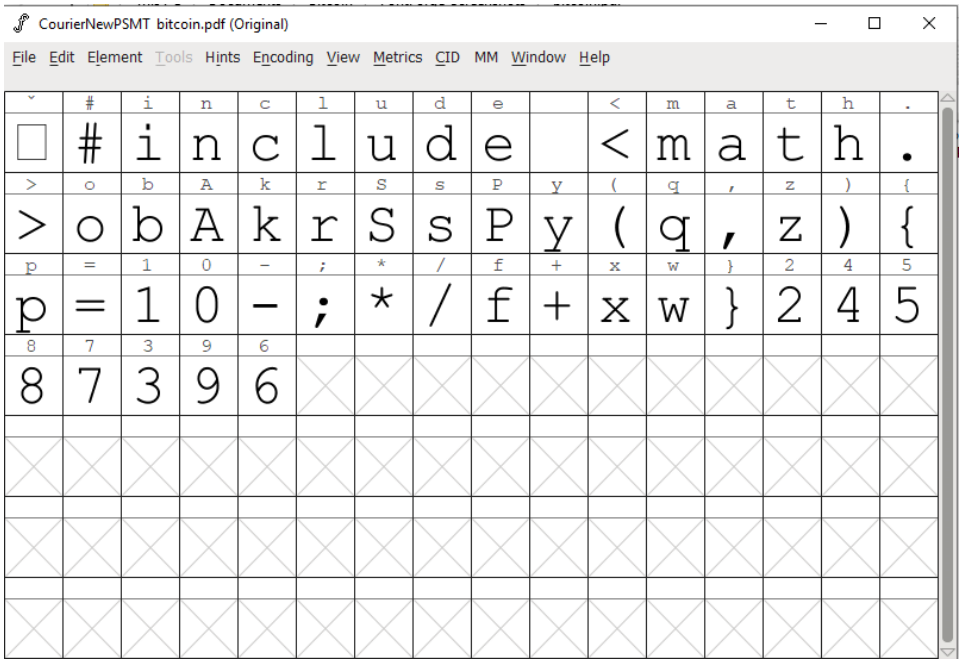
CenturySchoolbook-Bold



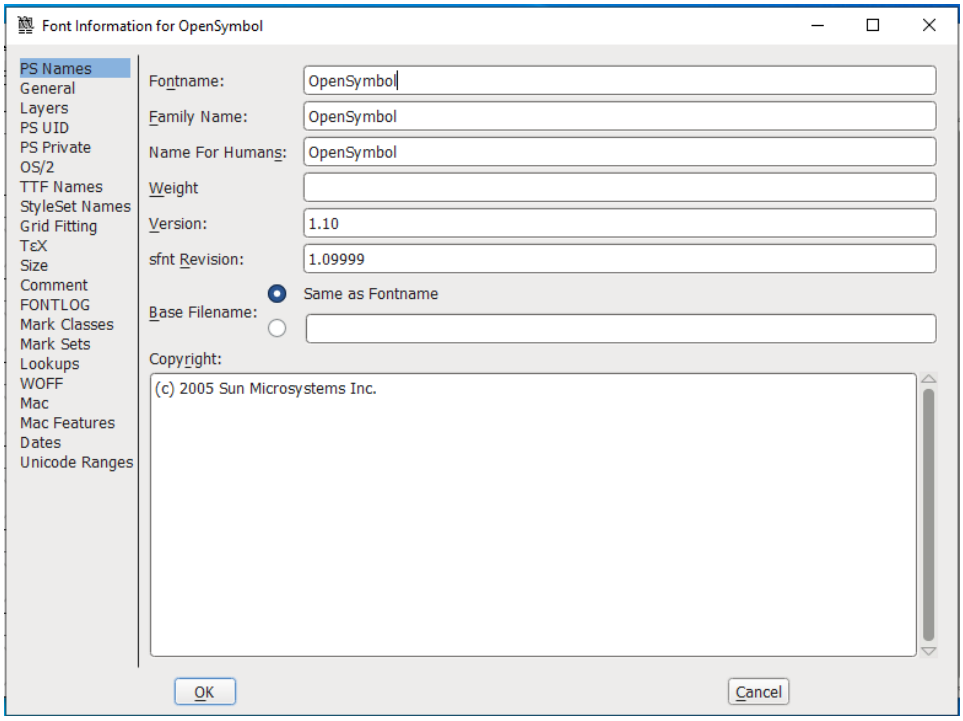


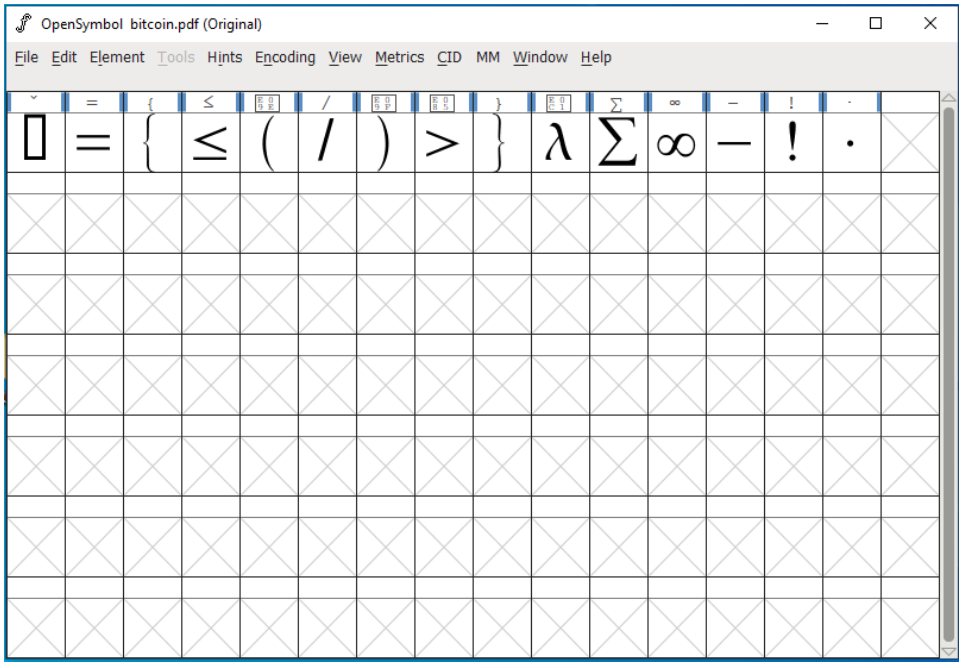
CourierNewPSMT



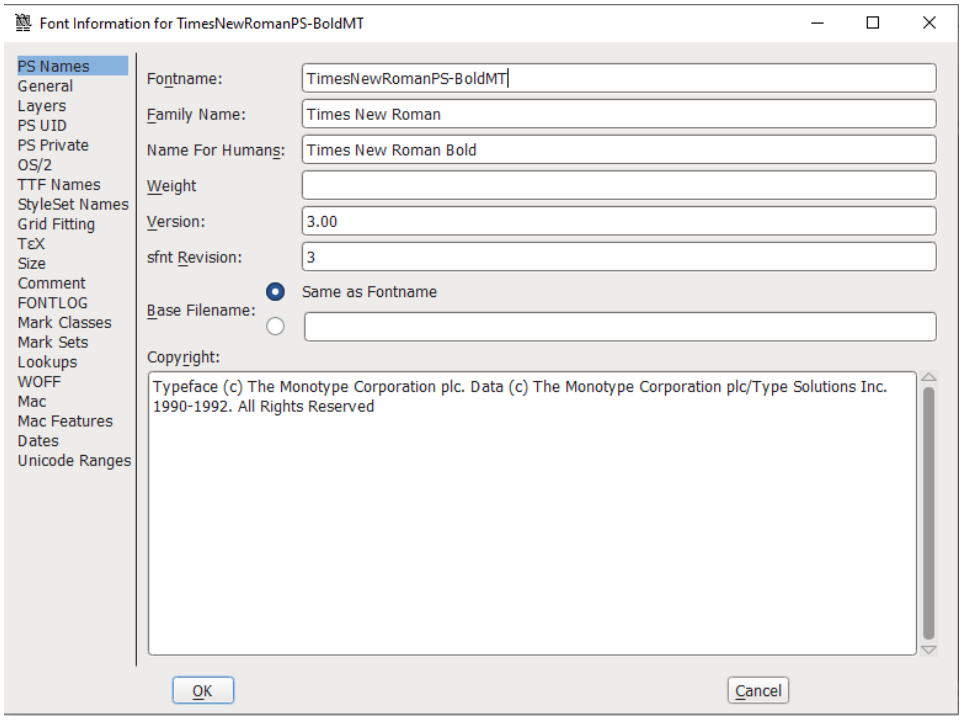


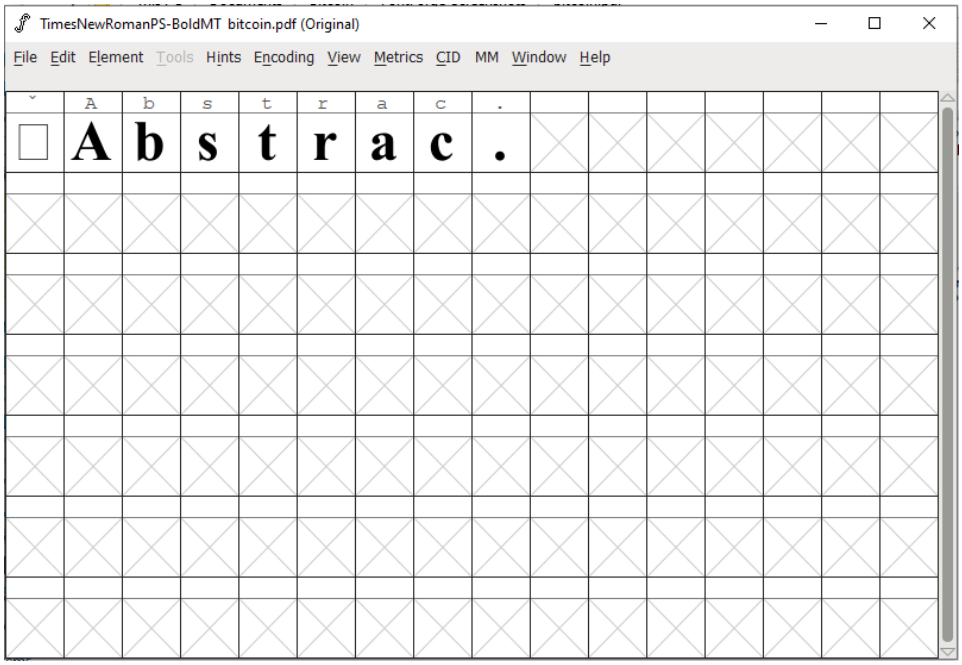
OpenSymbol



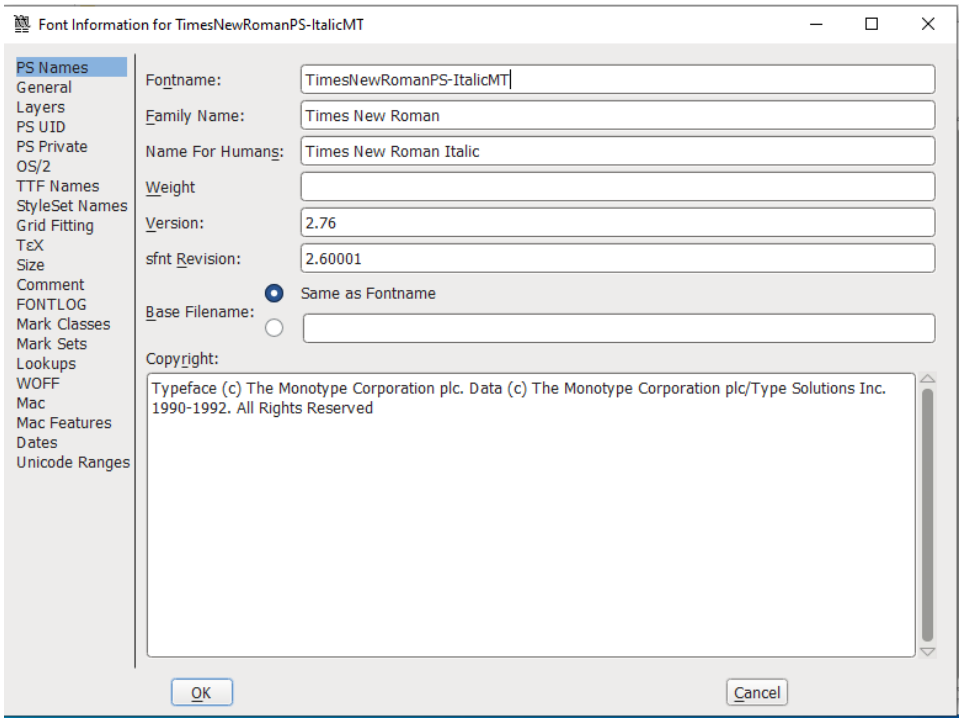


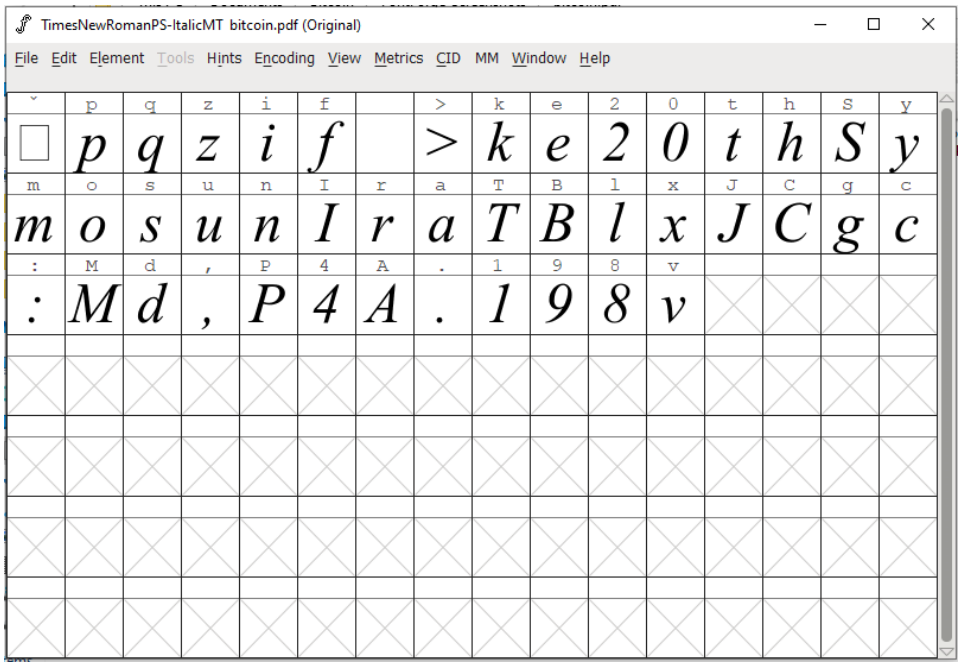
TimesNewRomanPS-BoldMT



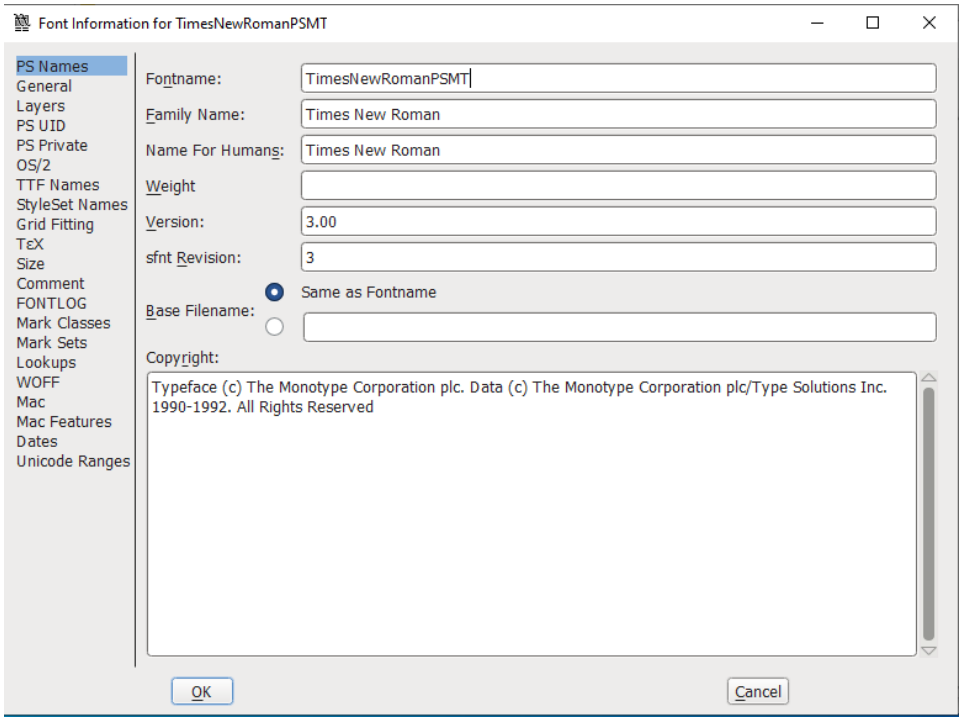


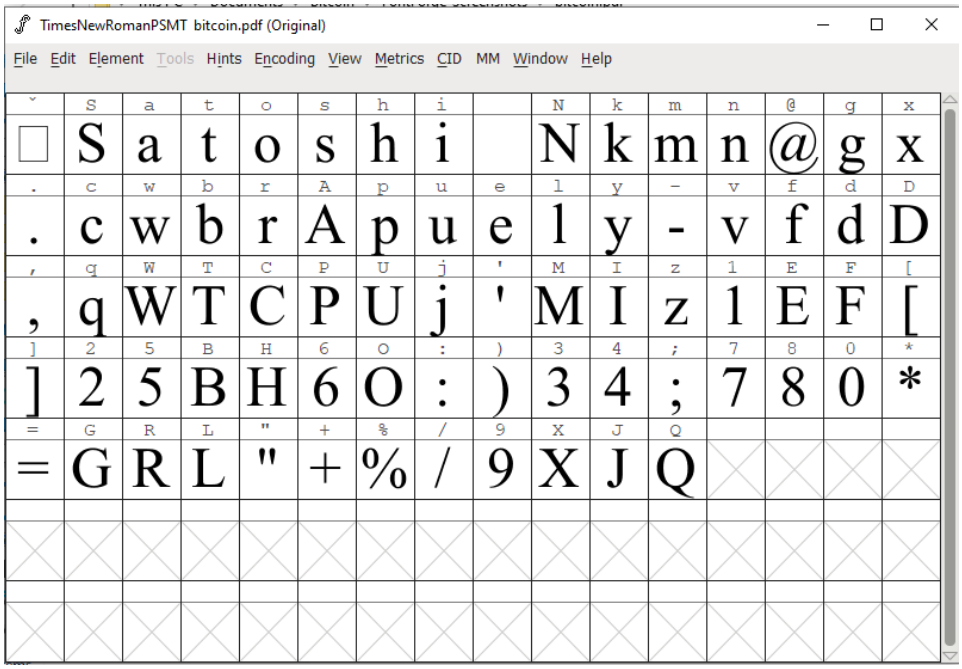
TimesNewRomanPS-ItalicMT





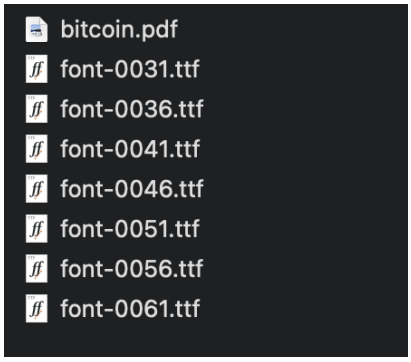
TimesNewRomanPSMT





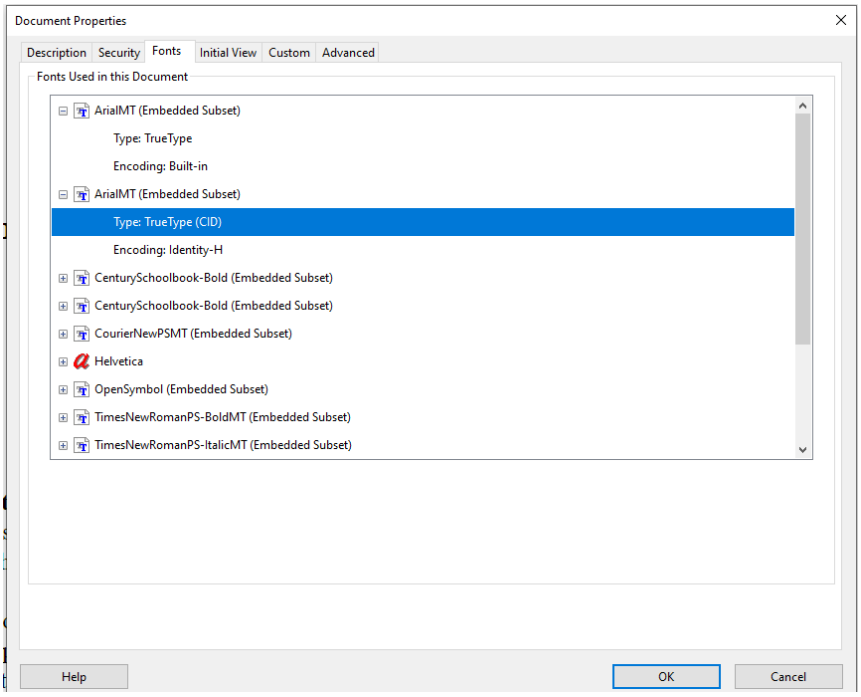
4.15.4 Skjermbilder fra «mutool extract» i MuPDF

```
emildaboda@Emils-MacBook-Pro bitcoin % mutool extract bitcoin.pdf
extracting font-0031.ttf
extracting font-0036.ttf
extracting font-0041.ttf
extracting font-0046.ttf
extracting font-0051.ttf
extracting font-0056.ttf
extracting font-0061.ttf
```

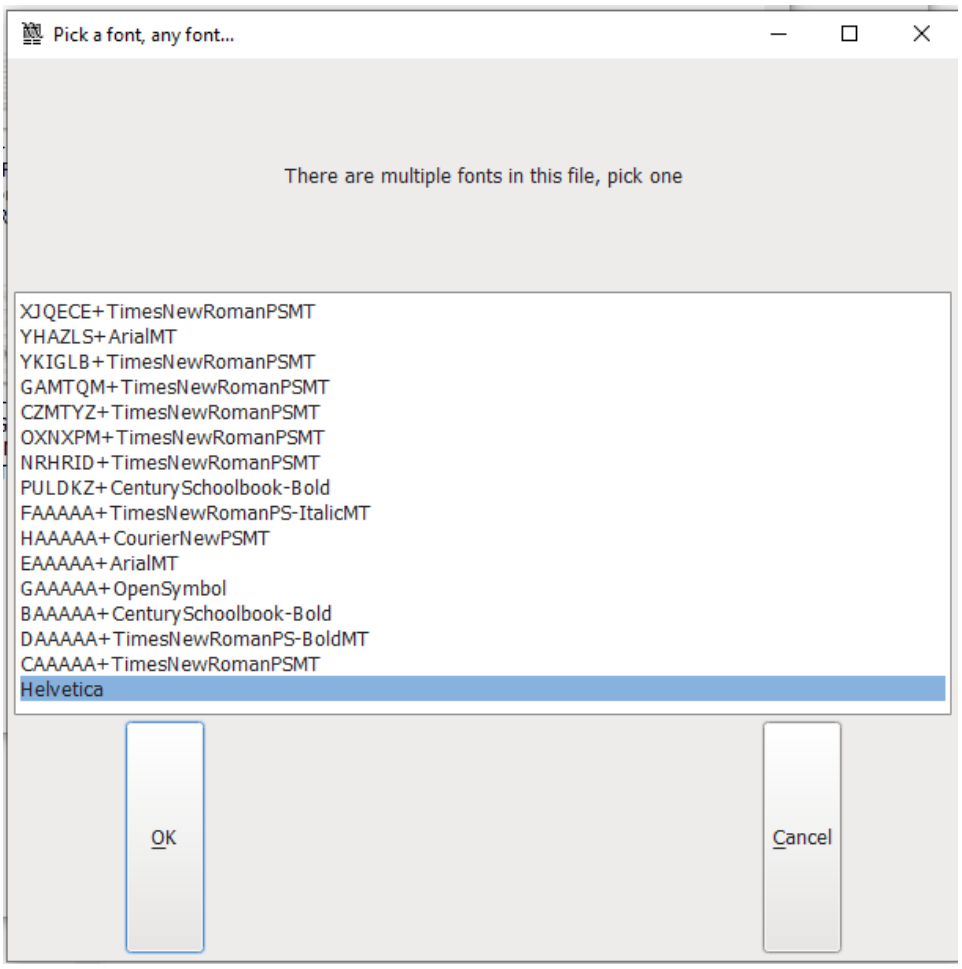


4.16 Vedlegg 16: Innholdsanalyse «SSRN-id3440802.pdf»

4.16.1 Fontoversikt

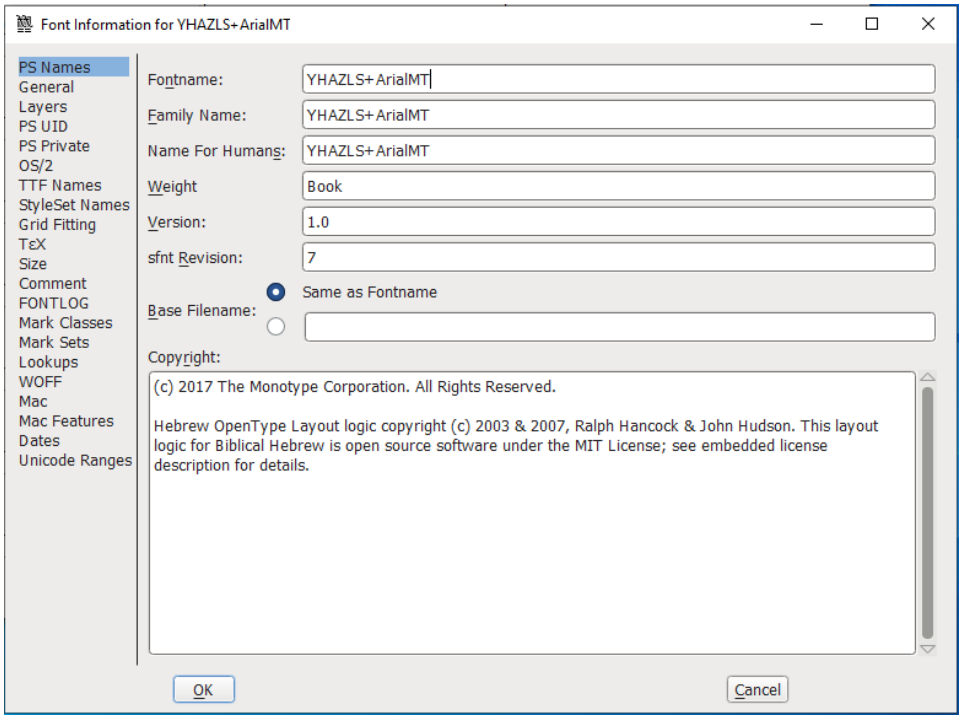


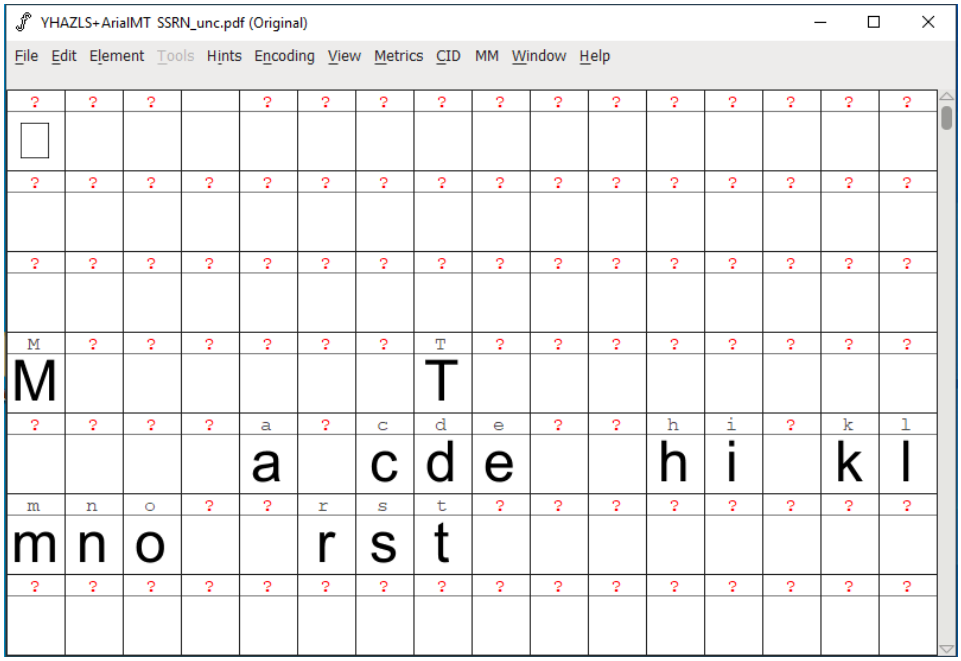
4.16.2 Font-metadata fra FontForge



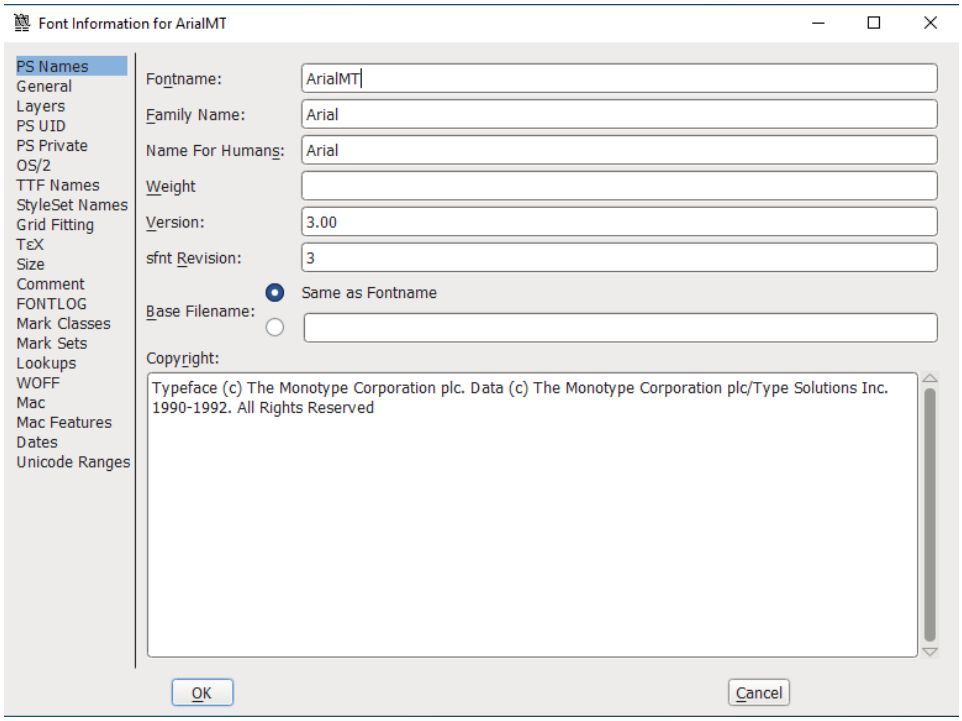
*Fontfilene for «XJQECE+TimesNewRomanPSMT», «YKIGLB+TimesNewRomanPSMT", "CZMTYZ+TimesNewRomanPSMT og «GAMTQM+TimesNewRomanPSMT» er identiske, og gjengis i FontForge som «GAMTQM+TimesNewRomanPSMT». KPMG har derfor inkludert metadata for disse fontfilene kun én gang under «GAMTQM+TimesNewRomanPSMT» i vedleggene som følger.

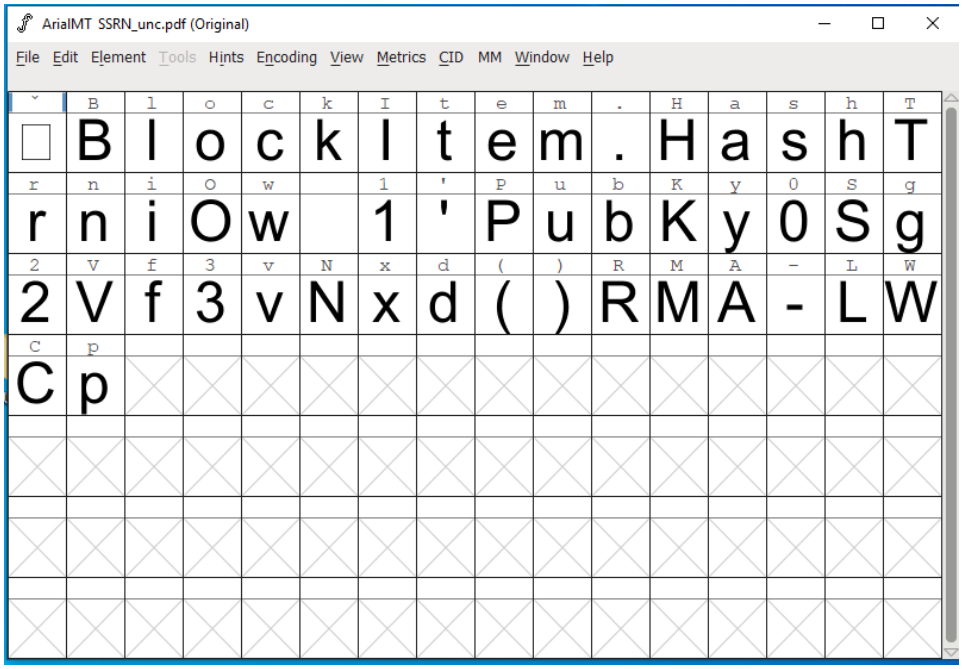
YHAZLS+ ArialMT



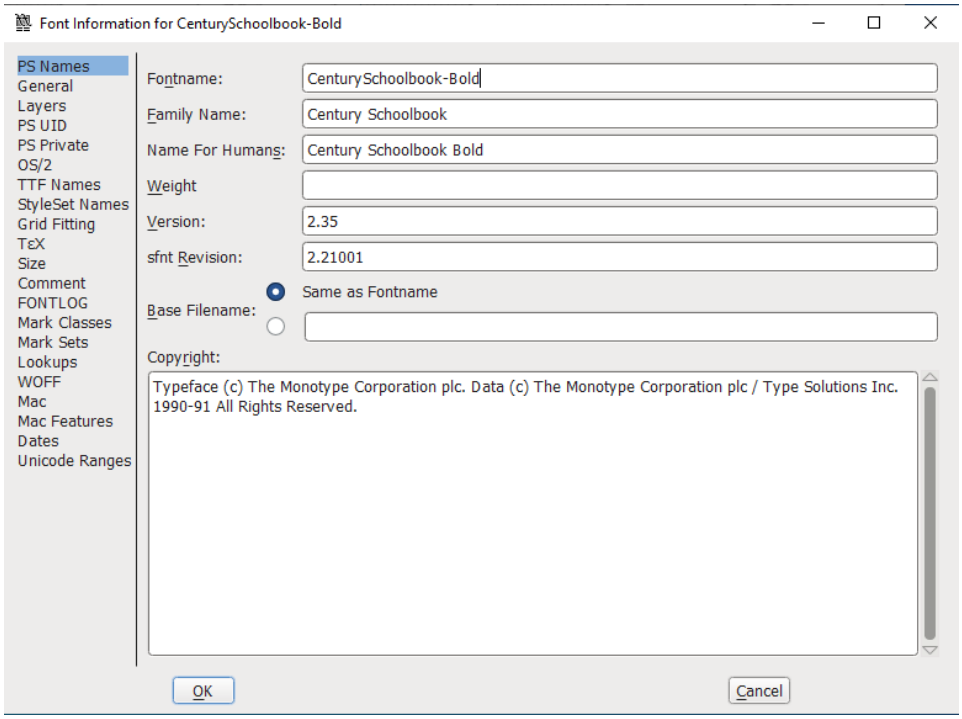


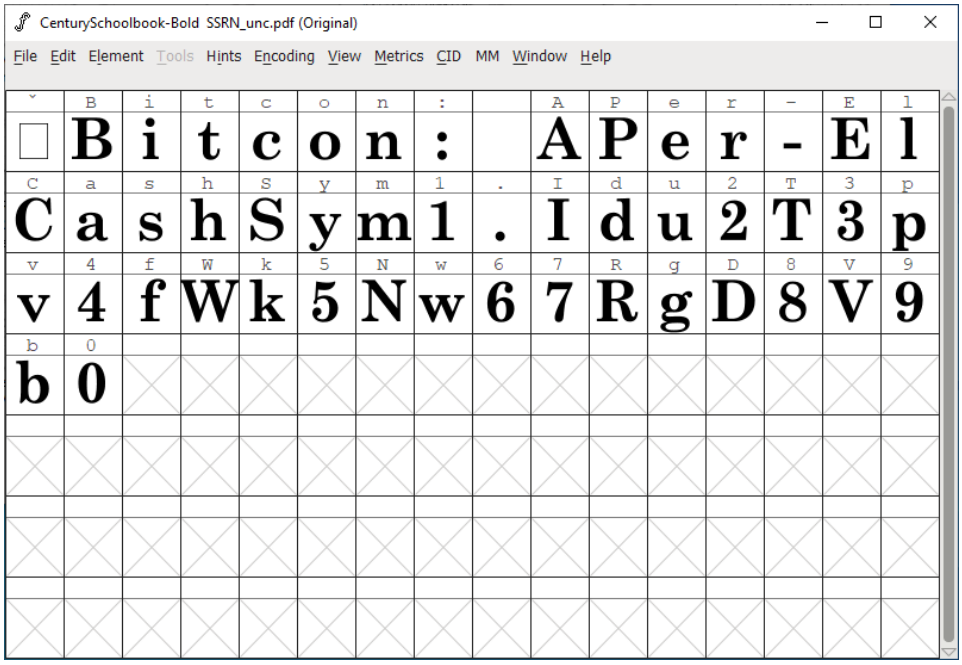
ArialMT



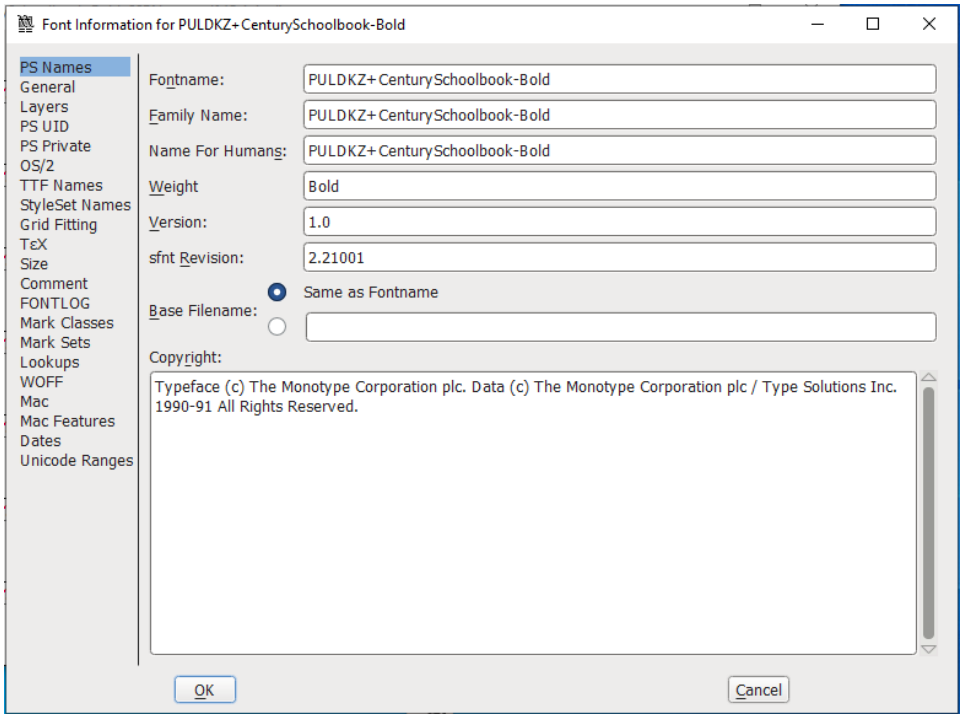


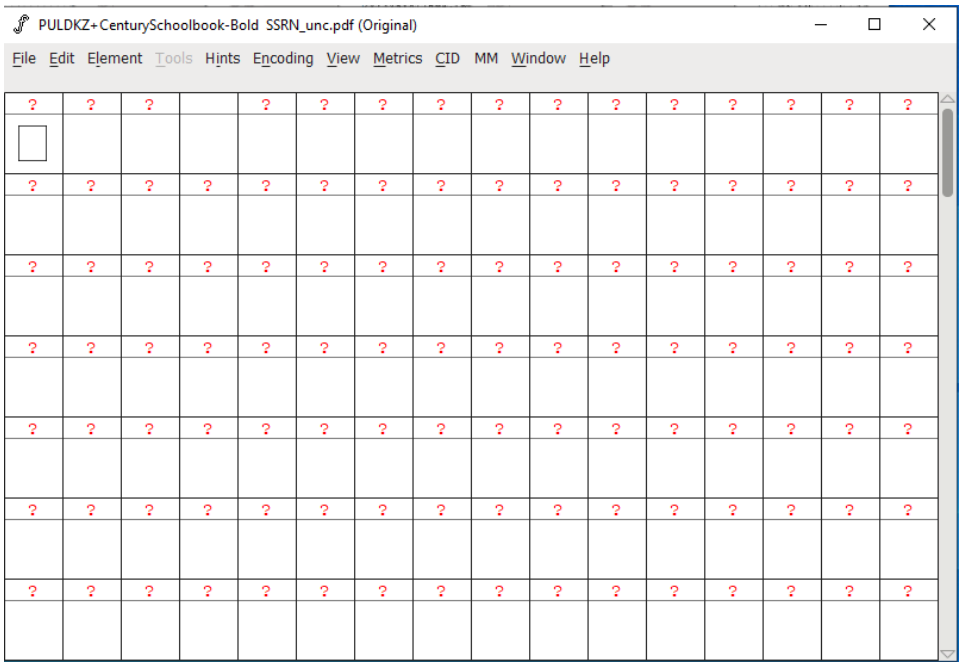
CenturySchoolbook-Bold



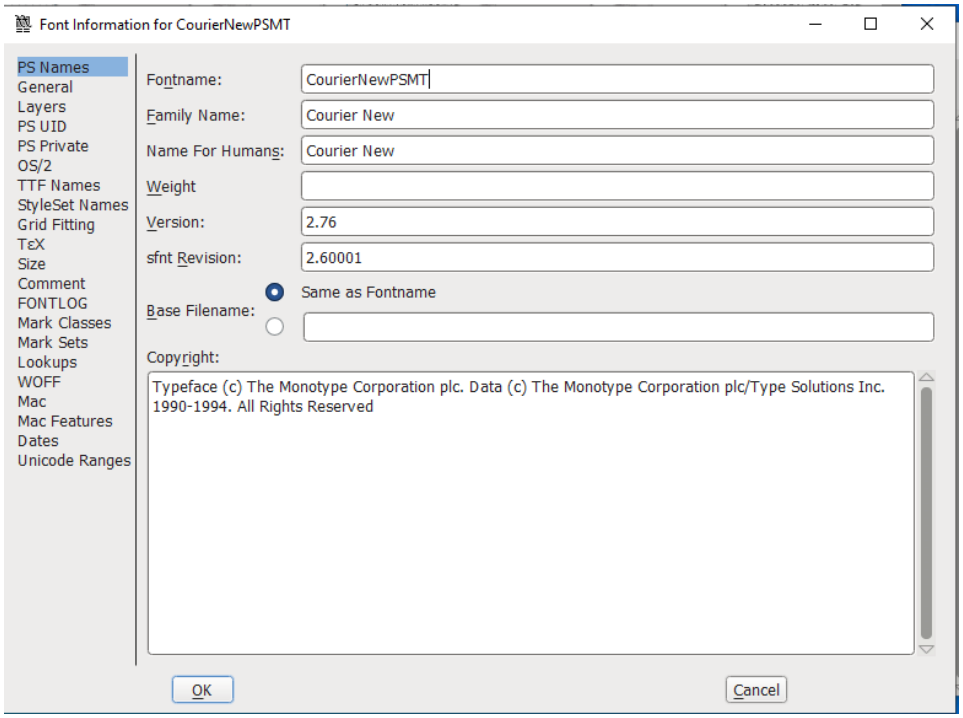


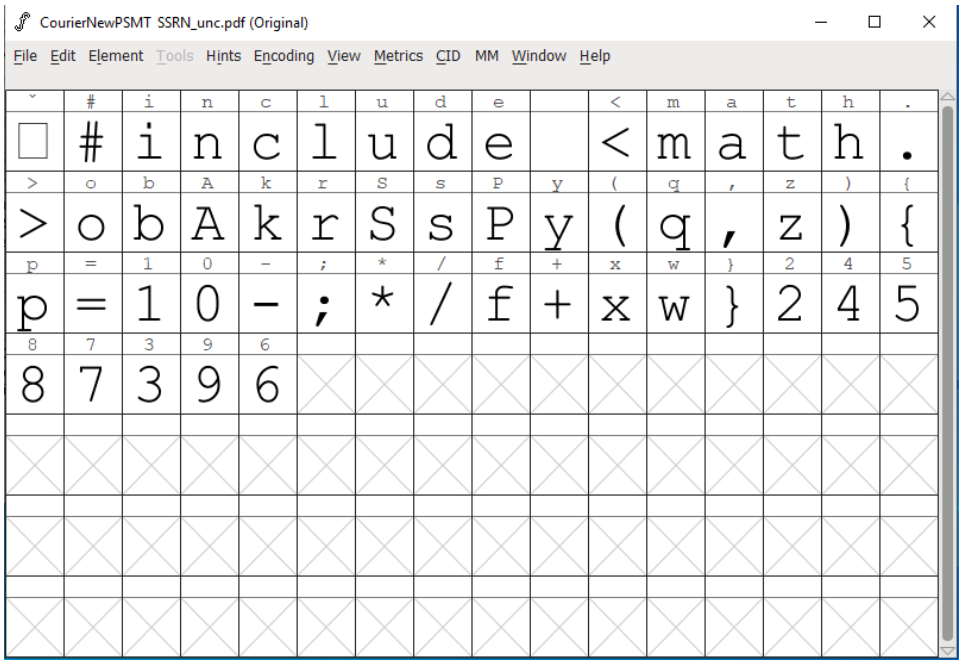
PULKDZ+ CenturySchoolbook-Bold



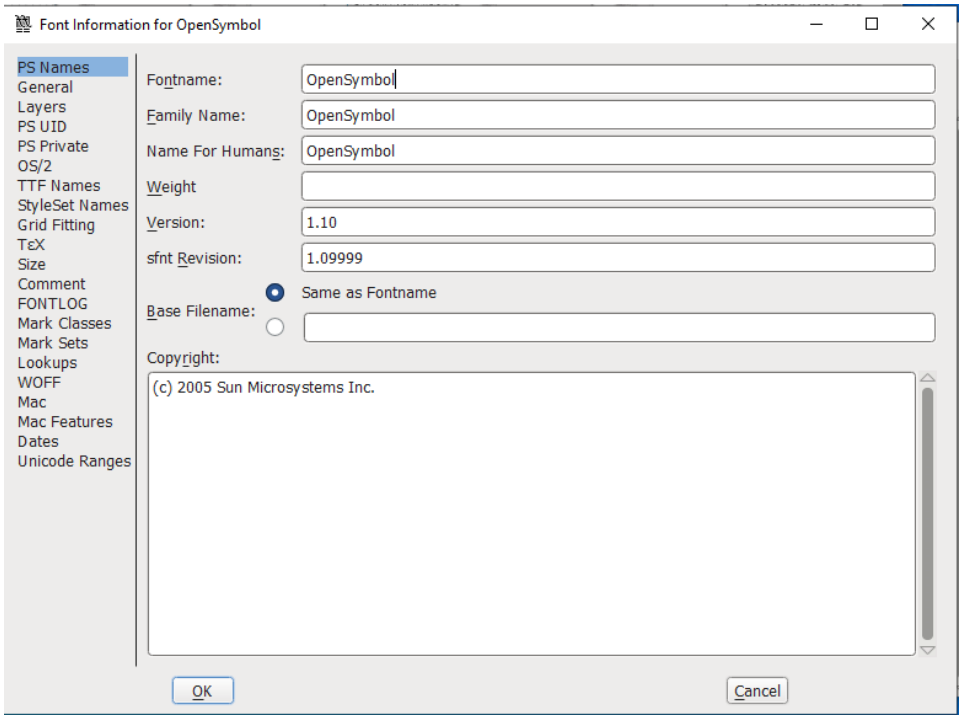


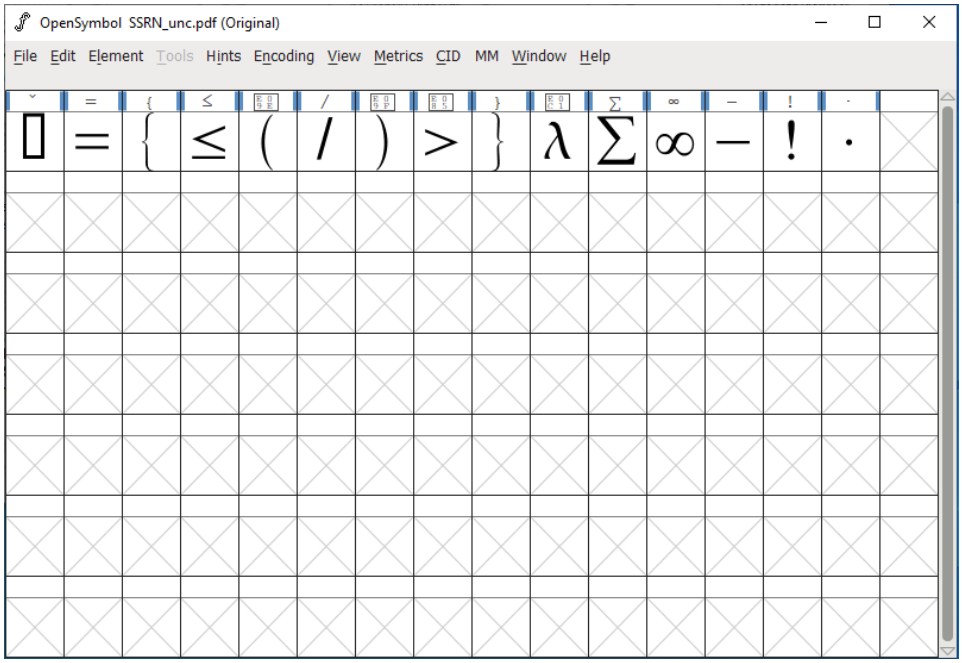
CourierNewPSMT



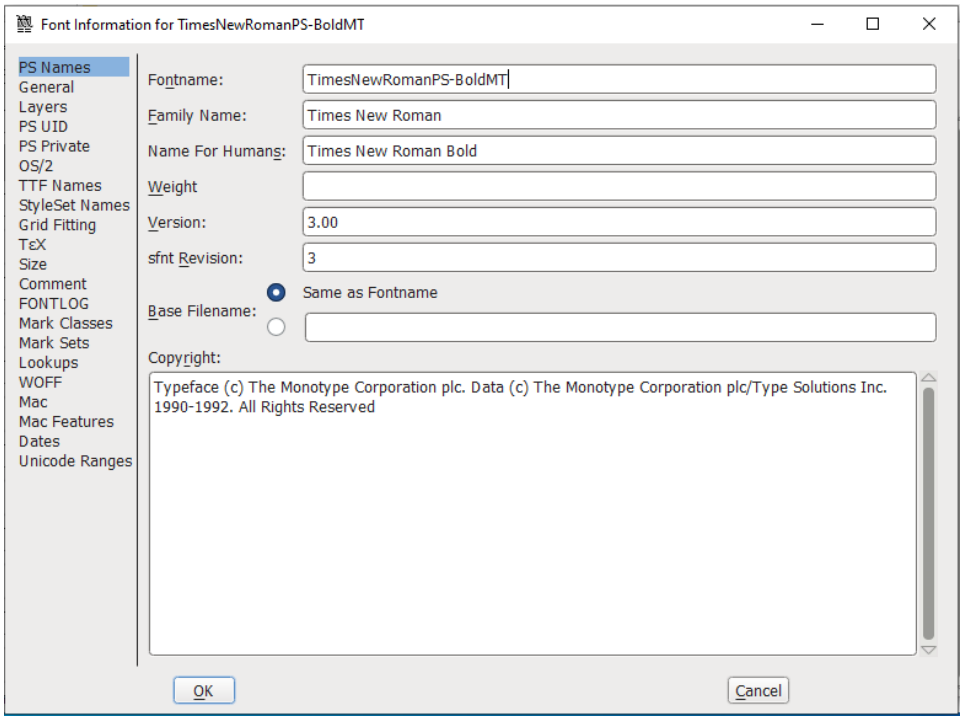


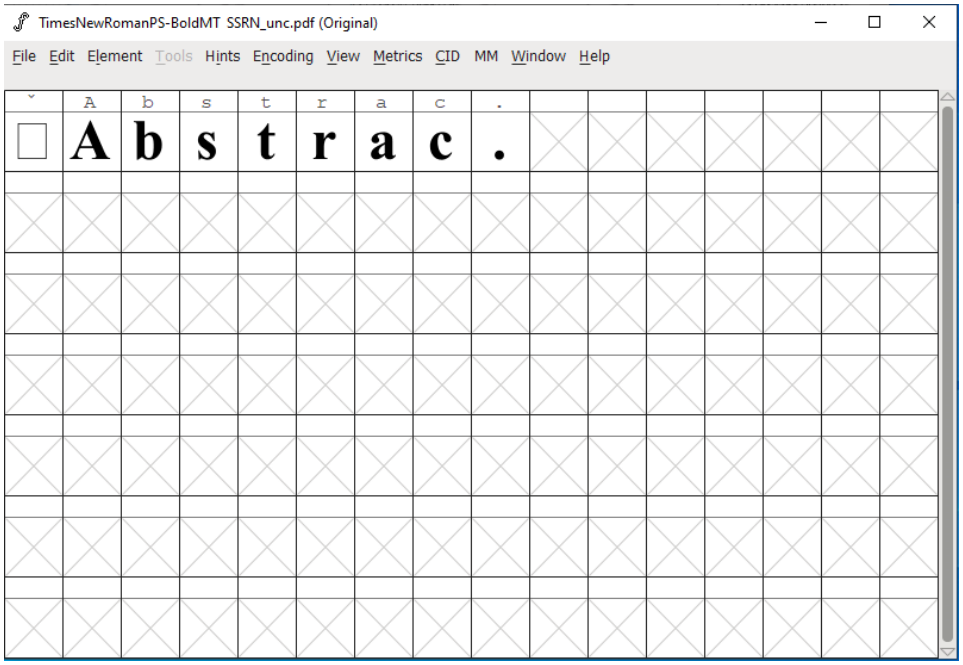
OpenSymbol



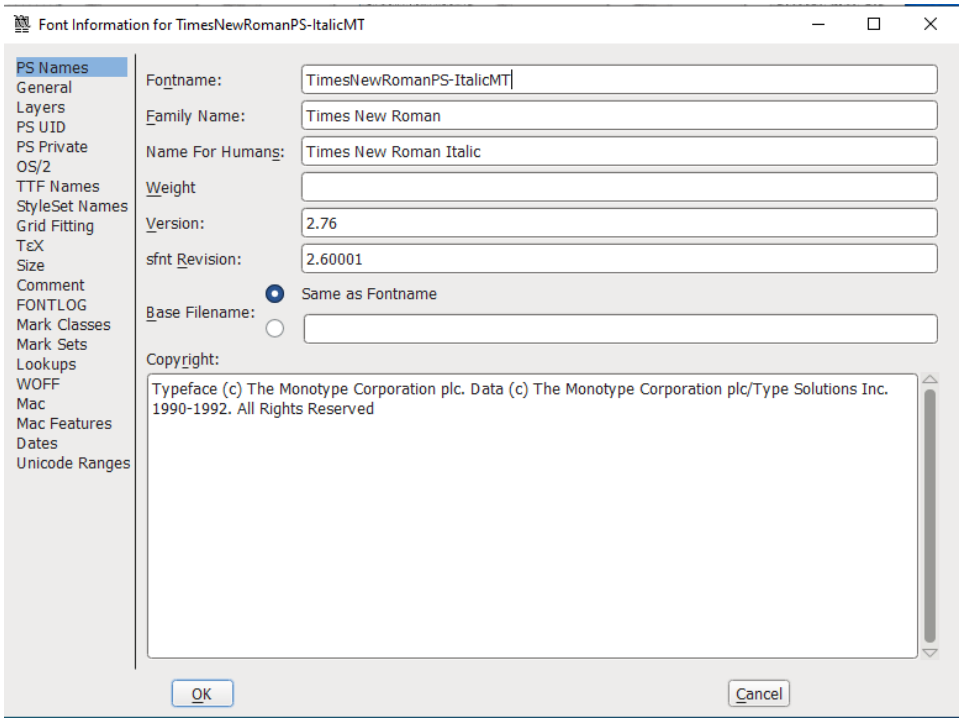


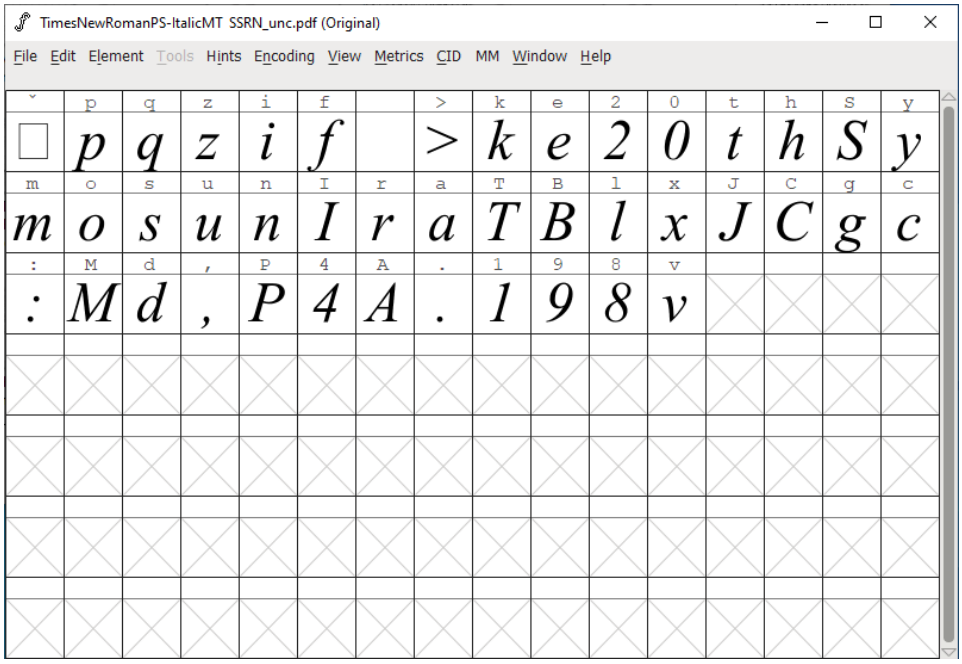
TimesNewRomanPS-BoldMT



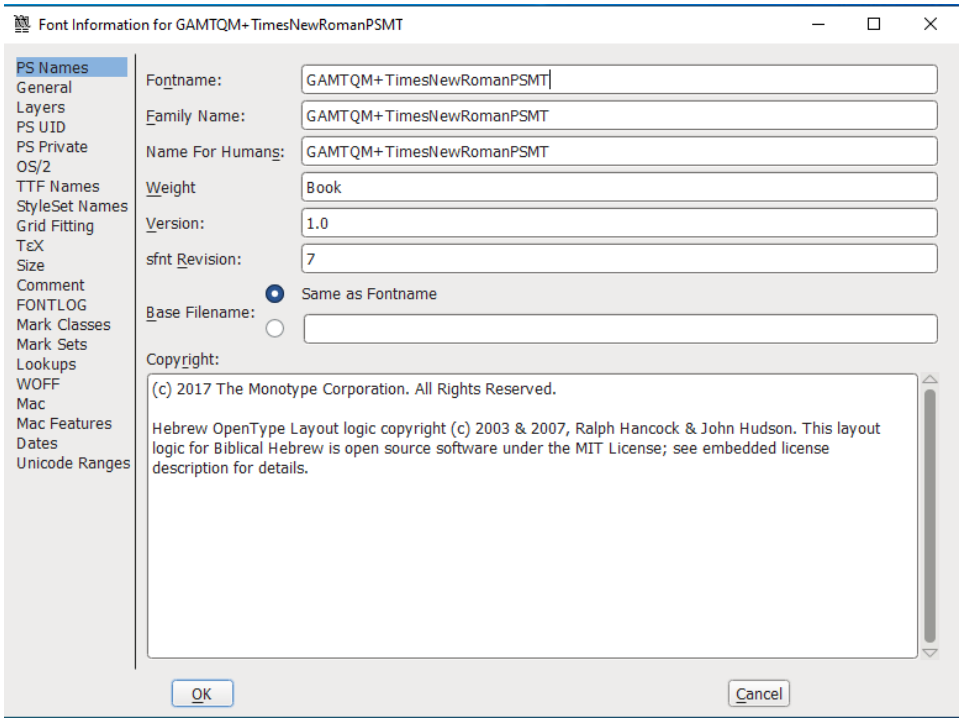


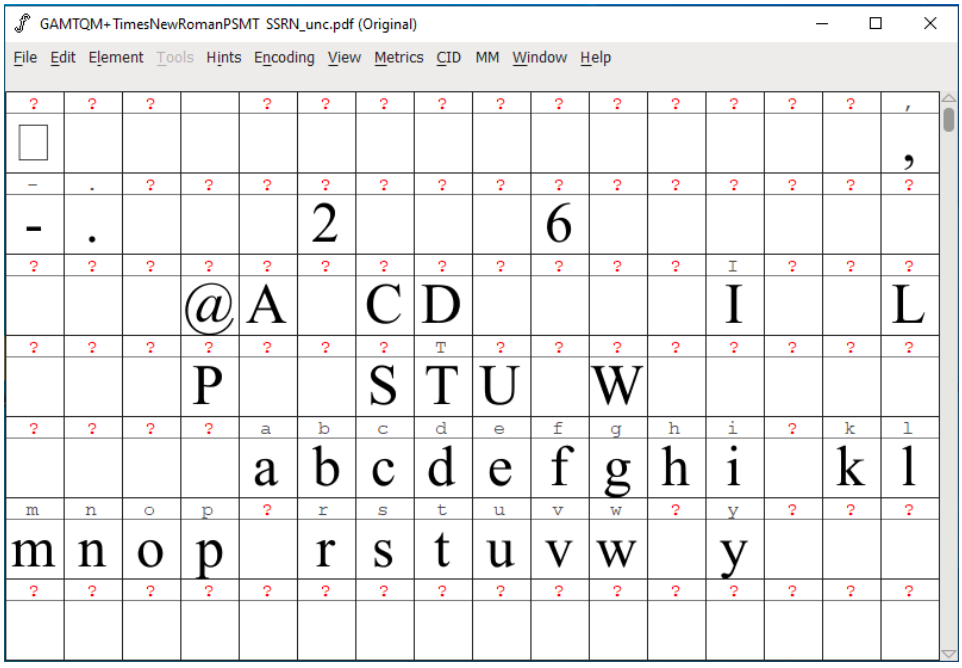
TimesNewRomanPS-ItalicMT



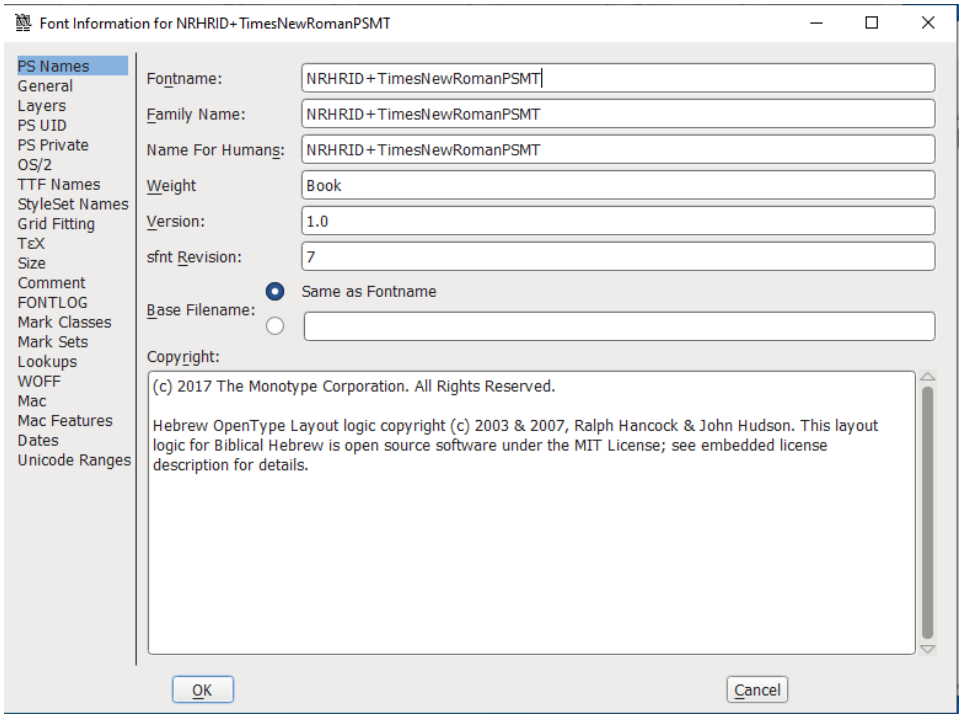


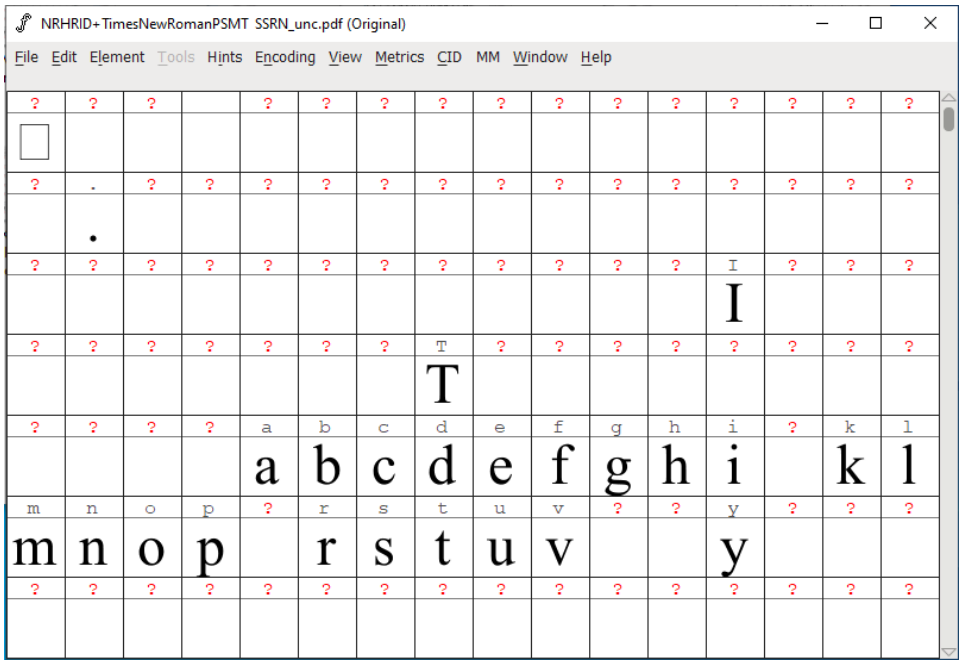
GAMTQM+TimesNewRomanPSMT



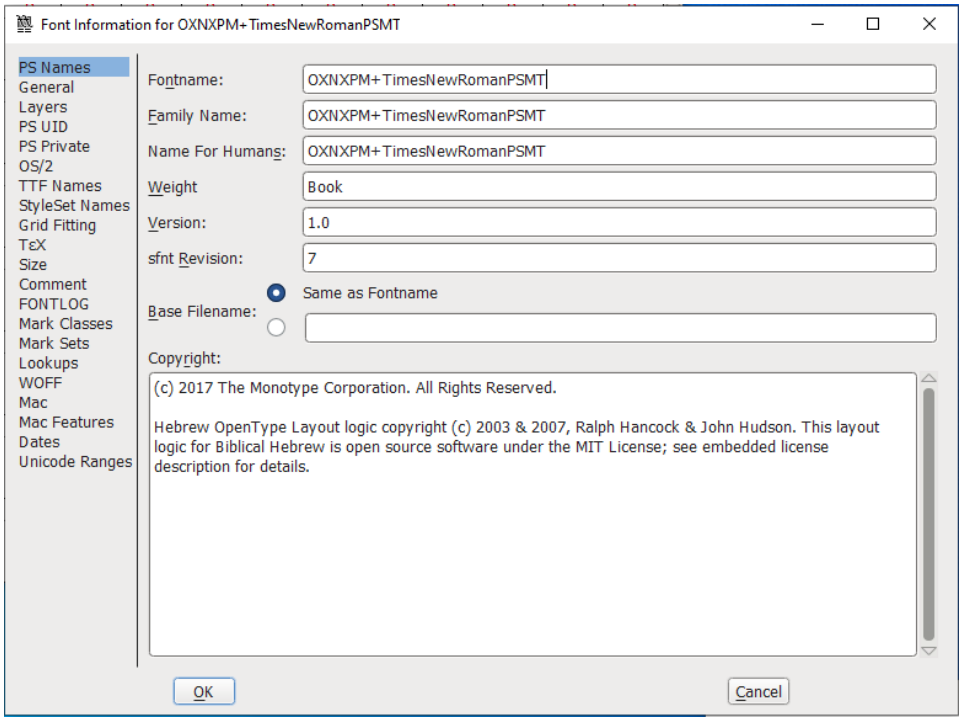


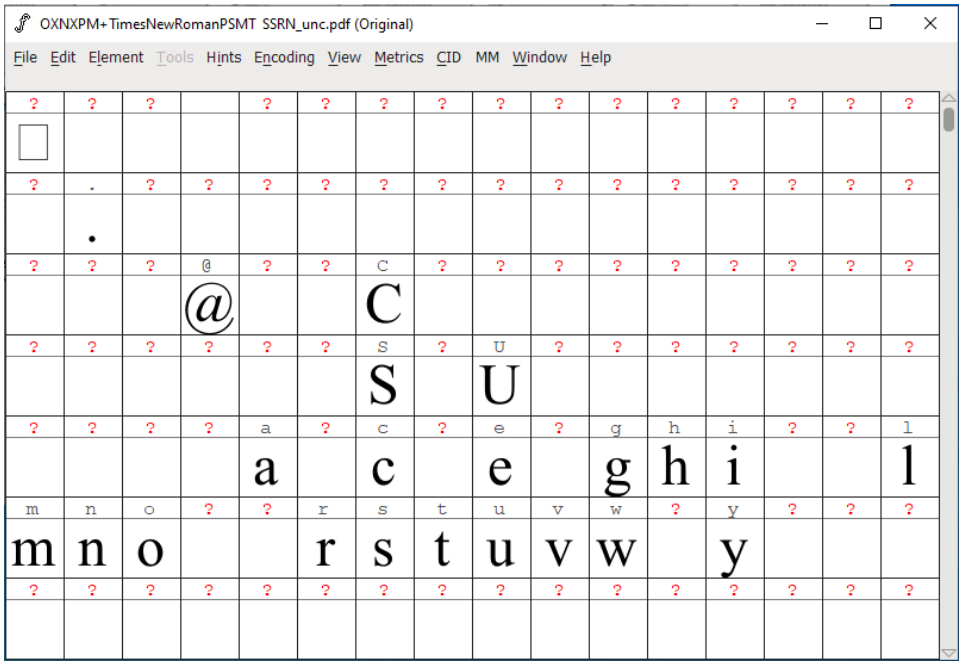
NRHRID+TimesNewRomanPSMT



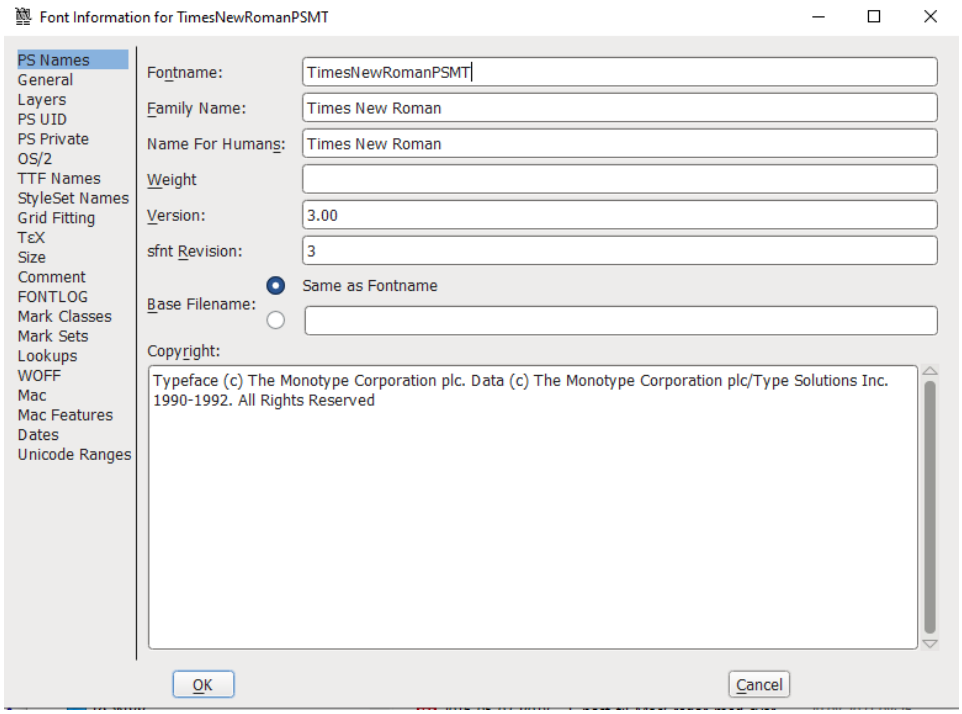


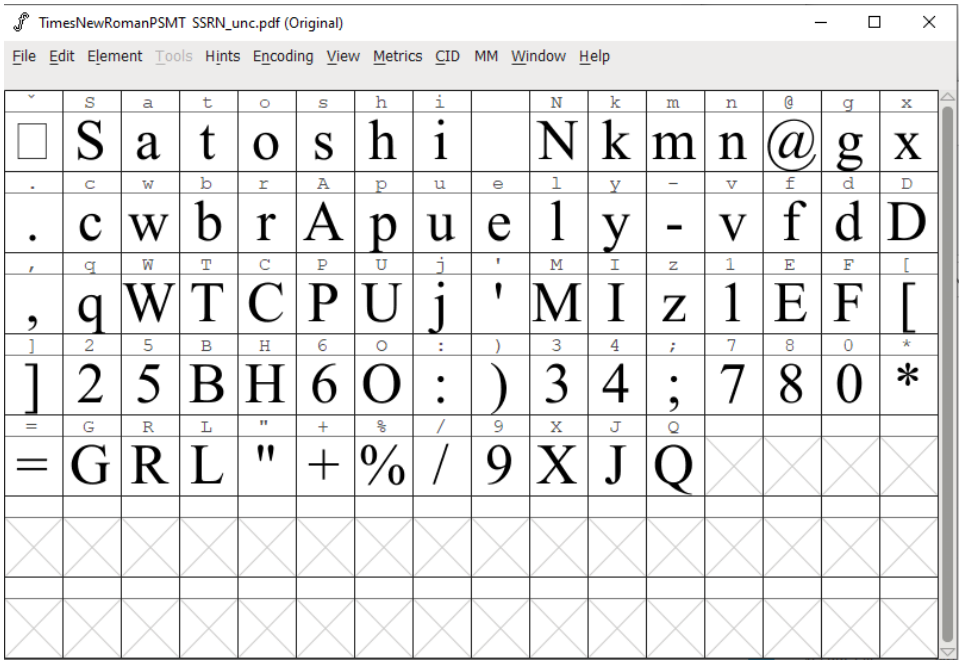
OXNXPM+TimesNewRomanPSMT



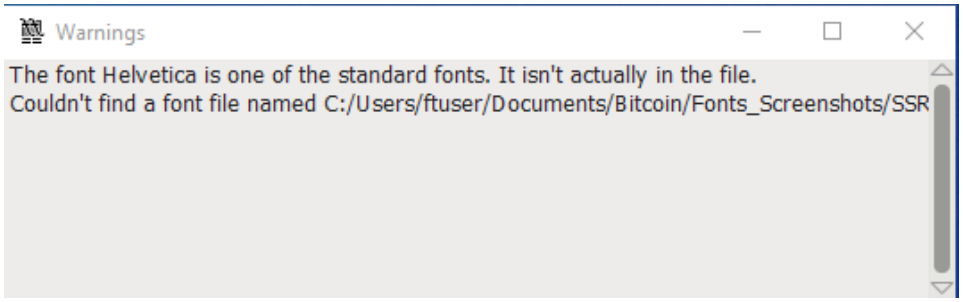


TimesNewRomanPSMT



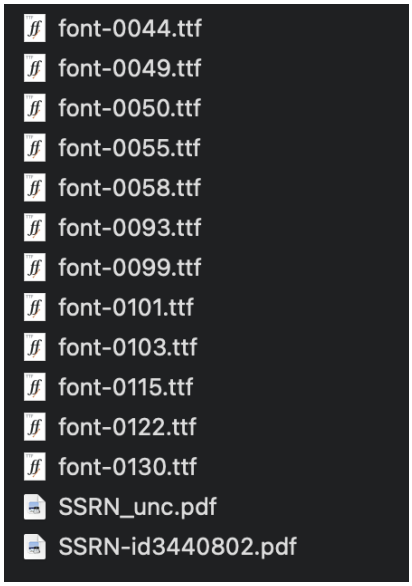


Helvetica



4.16.3 Skjermbilder fra «mutool extract» i MuPDF

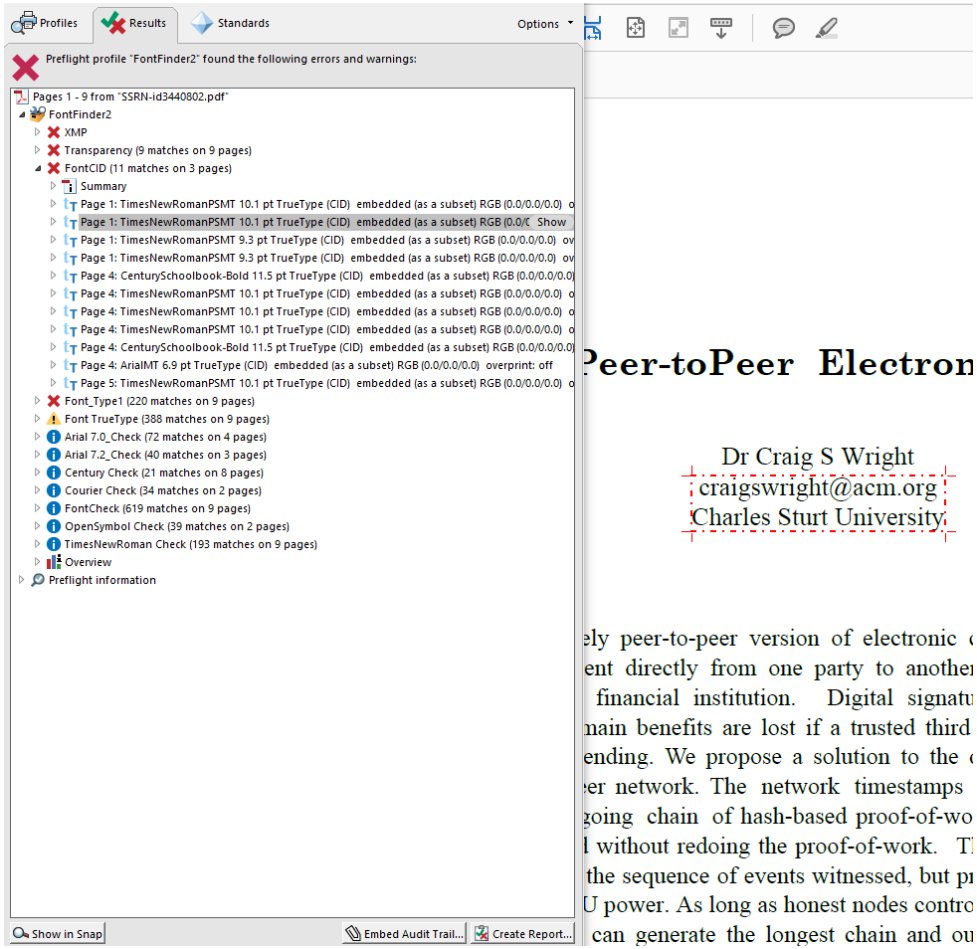
```
emildaboda@Emils-MacBook-Pro Bitcoin % mutool extract SSRN-id3440802.pdf
extracting font-0044.ttf
extracting font-0049.ttf
extracting font-0050.ttf
extracting font-0055.ttf
extracting font-0058.ttf
extracting font-0093.ttf
extracting font-0099.ttf
extracting font-0101.ttf
extracting font-0103.ttf
extracting font-0115.ttf
extracting font-0122.ttf
extracting font-0130.ttf
```



1303

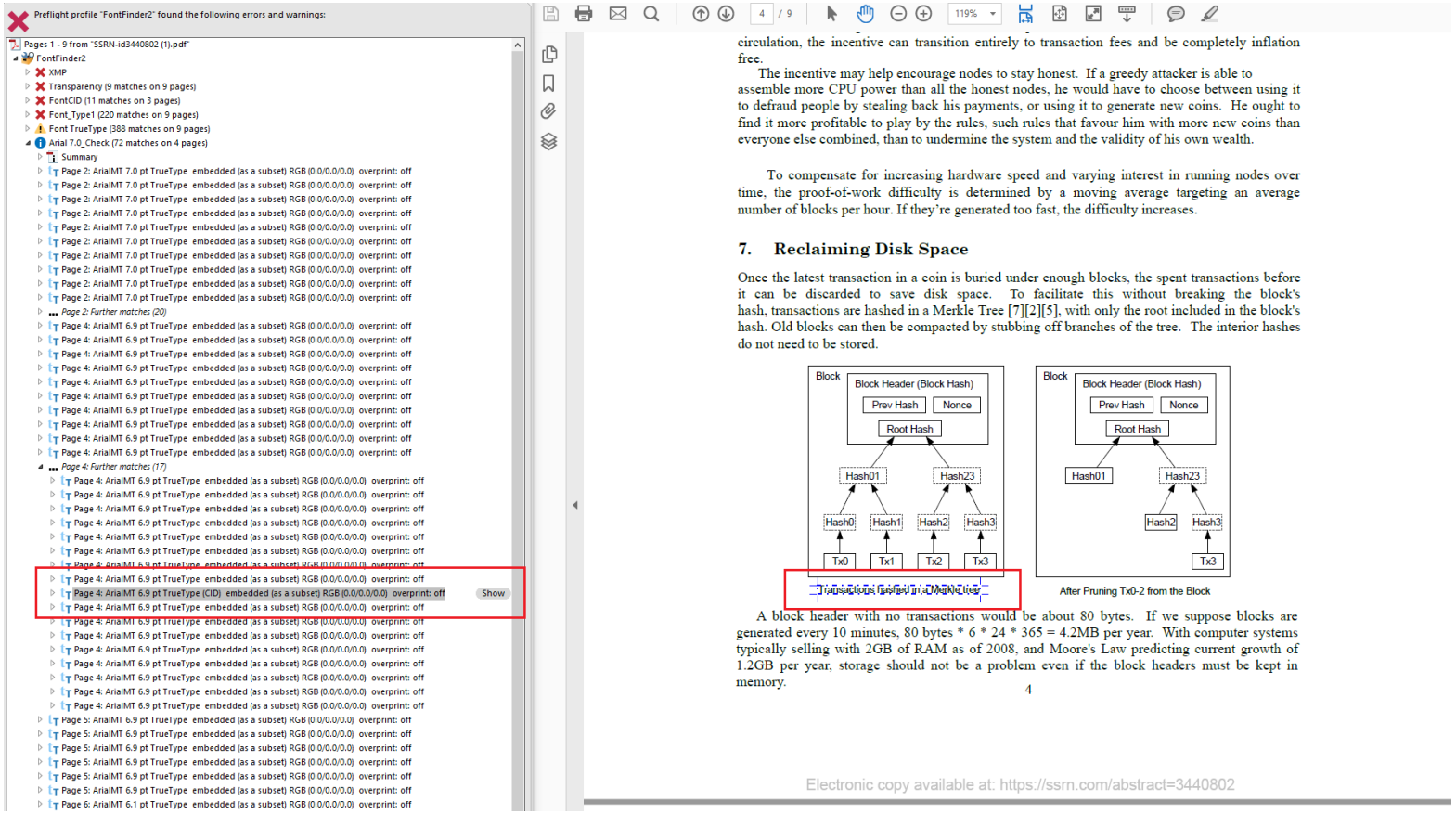
4.16.4 Font-type

Side 1:



1303

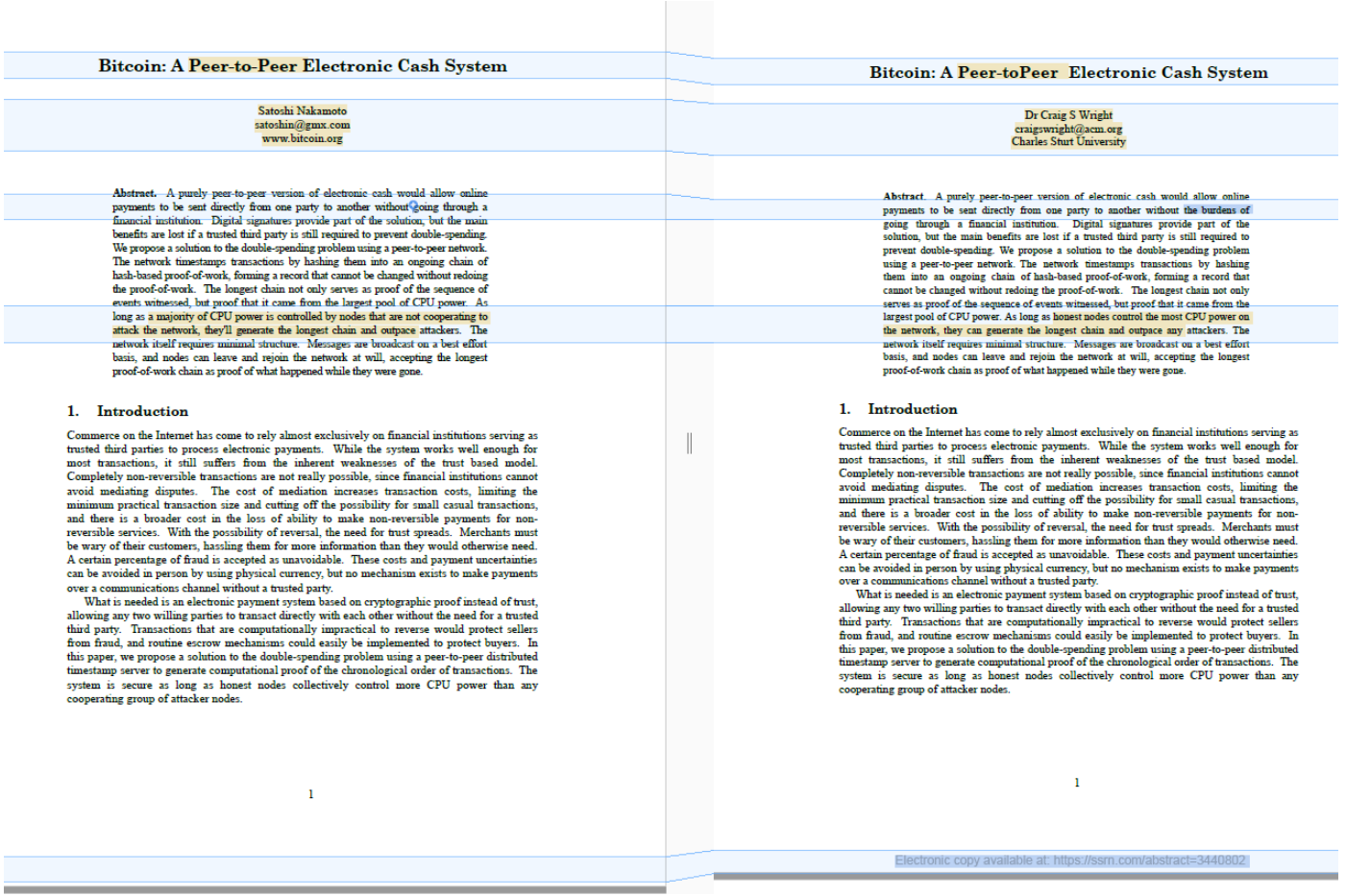
Side 4:



4.16.5 Tekstforskjeller fra «bitcoin.pdf» vs. «SSRN-id3440802.pdf»

All «endret» tekst er uthevet i gult, all tillegg av tekst er uthevet i blått og andre endringer er markert med et hvitt kryss på rød bakgrunn.

Side 1:



Bitcoin: A Peer-to-Peer Electronic Cash System

Satoshi Nakamoto
satoshin@gmx.com
www.bitcoin.org

Abstract. A purely peer-to-peer version of electronic cash would allow online payments to be sent directly from one party to another without going through a financial institution. Digital signatures provide part of the solution, but the main benefits are lost if a trusted third party is still required to prevent double-spending. We propose a solution to the double-spending problem using a peer-to-peer network. The network timestamps transactions by hashing them into an ongoing chain of hash-based proof-of-work, forming a record that cannot be changed without redoing the proof-of-work. The longest chain not only serves as proof of the sequence of events witnessed, but proof that it came from the largest pool of CPU power. As long as a majority of CPU power is controlled by nodes that are not cooperating to attack the network, they'll generate the longest chain and outpace attackers. The network itself requires minimal structure. Messages are broadcast on a best effort basis, and nodes can leave and rejoin the network at will, accepting the longest proof-of-work chain as proof of what happened while they were gone.

1. Introduction

Commerce on the Internet has come to rely almost exclusively on financial institutions serving as trusted third parties to process electronic payments. While the system works well enough for most transactions, it still suffers from the inherent weaknesses of the trust based model. Completely non-reversible transactions are not really possible, since financial institutions cannot avoid mediating disputes. The cost of mediation increases transaction costs, limiting the minimum practical transaction size and cutting off the possibility for small casual transactions, and there is a broader cost in the loss of ability to make non-reversible payments for non-reversible services. With the possibility of reversal, the need for trust spreads. Merchants must be wary of their customers, hassling them for more information than they would otherwise need. A certain percentage of fraud is accepted as unavoidable. These costs and payment uncertainties can be avoided in person by using physical currency, but no mechanism exists to make payments over a communications channel without a trusted party.

What is needed is an electronic payment system based on cryptographic proof instead of trust, allowing any two willing parties to transact directly with each other without the need for a trusted third party. Transactions that are computationally impractical to reverse would protect sellers from fraud, and routine escrow mechanisms could easily be implemented to protect buyers. In this paper, we propose a solution to the double-spending problem using a peer-to-peer distributed timestamp server to generate computational proof of the chronological order of transactions. The system is secure as long as honest nodes collectively control more CPU power than any cooperating group of attacker nodes.

1

Bitcoin: A Peer-toPeer Electronic Cash System

Dr Craig S Wright
craigswright@acm.org
Charles Sturt University

Abstract. A purely peer-to-peer version of electronic cash would allow online payments to be sent directly from one party to another without the burdens of going through a financial institution. Digital signatures provide part of the solution, but the main benefits are lost if a trusted third party is still required to prevent double-spending. We propose a solution to the double-spending problem using a peer-to-peer network. The network timestamps transactions by hashing them into an ongoing chain of hash-based proof-of-work, forming a record that cannot be changed without redoing the proof-of-work. The longest chain not only serves as proof of the sequence of events witnessed, but proof that it came from the largest pool of CPU power. As long as honest nodes control the most CPU power on the network, they can generate the longest chain and outpace any attackers. The network itself requires minimal structure. Messages are broadcast on a best effort basis, and nodes can leave and rejoin the network at will, accepting the longest proof-of-work chain as proof of what happened while they were gone.

1. Introduction

Commerce on the Internet has come to rely almost exclusively on financial institutions serving as trusted third parties to process electronic payments. While the system works well enough for most transactions, it still suffers from the inherent weaknesses of the trust based model. Completely non-reversible transactions are not really possible, since financial institutions cannot avoid mediating disputes. The cost of mediation increases transaction costs, limiting the minimum practical transaction size and cutting off the possibility for small casual transactions, and there is a broader cost in the loss of ability to make non-reversible payments for non-reversible services. With the possibility of reversal, the need for trust spreads. Merchants must be wary of their customers, hassling them for more information than they would otherwise need. A certain percentage of fraud is accepted as unavoidable. These costs and payment uncertainties can be avoided in person by using physical currency, but no mechanism exists to make payments over a communications channel without a trusted party.

What is needed is an electronic payment system based on cryptographic proof instead of trust, allowing any two willing parties to transact directly with each other without the need for a trusted third party. Transactions that are computationally impractical to reverse would protect sellers from fraud, and routine escrow mechanisms could easily be implemented to protect buyers. In this paper, we propose a solution to the double-spending problem using a peer-to-peer distributed timestamp server to generate computational proof of the chronological order of transactions. The system is secure as long as honest nodes collectively control more CPU power than any cooperating group of attacker nodes.

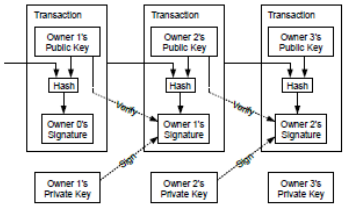
1

1306

Side 2:

2. Transactions

We define an electronic coin as a chain of digital signatures. Each owner transfers the coin to the next by digitally signing a hash of the previous transaction and the public key of the next owner and adding these to the end of the coin. A payee can verify the signatures to verify the chain of ownership.

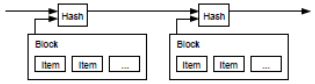


The problem of course is the payee can't verify that one of the owners did not double-spend the coin. A common solution is to introduce a trusted central authority, or mint, that checks every transaction for double spending. After each transaction, the coin must be returned to the mint to issue a new coin, and only coins issued directly from the mint are trusted not to be double-spent. The problem with this solution is that the fate of the entire money system depends on the company running the mint, with every transaction having to go through them, just like a bank.

We need a way for the payee to know that the previous owners did not sign any earlier transactions. For our purposes, the earliest transaction is the one that counts, so we don't care about later attempts to double-spend. The only way to confirm the absence of a transaction is to be aware of all transactions. In the mint based model, the mint was aware of all transactions and decided which arrived first. To accomplish this without a trusted party, transactions must be publicly announced [1], and we need a system for participants to agree on a single history of the order in which they were received. The payee needs proof that at the time of each transaction, the majority of nodes agreed it was the first received.

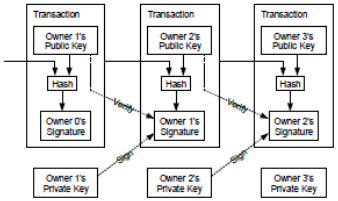
3. Timestamp Server

The solution we propose begins with a timestamp server. A timestamp server works by taking a hash of a block of items to be timestamped and widely publishing the hash, such as in a newspaper or Usenet post [2-5]. The timestamp proves that the data must have existed at the time, obviously, in order to get into the hash. Each timestamp includes the previous timestamp in its hash, forming a chain, with each additional timestamp reinforcing the ones before it.



2. Transactions

We define an electronic coin as a chain of digital signatures. Each owner transfers the coin to the next by digitally signing a hash of the previous transaction and the public key of the next owner and adding these to the end of the coin. A payee can verify the signatures to verify the chain of ownership.

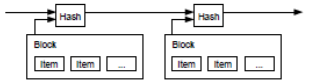


The problem of course is the payee can't verify that one of the owners did not double-spend the coin. A common solution is to introduce a trusted central authority, or mint, that checks every transaction for double spending. After each transaction, the coin must be returned to the mint to issue a new coin, and only coins issued directly from the mint are trusted not to be double-spent. The problem with this solution is that the fate of the entire money system depends on the company running the mint, with every transaction having to go through them, just like a bank.

We need a way for the payee to know that the previous owners did not sign any earlier transactions. For our purposes, the earliest transaction is the one that counts, so we don't care about later attempts to double-spend. The only way to confirm the absence of a transaction is to be aware of all transactions. In the mint based model, the mint was aware of all transactions and decided which arrived first. To accomplish this without a trusted party, transactions must be publicly announced [1], and we need a system for participants to agree on a single history of the order in which they were received. The payee needs proof that at the time of each transaction, the majority of nodes agreed it was the first received.

3. Timestamp Server

The solution we propose begins with a timestamp server. A timestamp server works by taking a hash of a block of items to be timestamped and widely publishing the hash, such as in a newspaper or Usenet post [2-5]. The timestamp proves that the data must have existed at the time, obviously, in order to get into the hash. Each timestamp includes the previous timestamp in its hash, forming a chain, with each additional timestamp reinforcing the ones before it.



1306

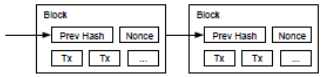
1307

Side 3:

4. Proof-of-Work

To implement a distributed timestamp server on a peer-to-peer basis, we will need to use a proof-of-work system similar to Adam Back's Hashcash [6], rather than newspaper or Usenet posts. The proof-of-work involves scanning for a value that when hashed, such as with SHA-256, the hash begins with a number of zero bits. The average work required is exponential in the number of zero bits required and can be verified by executing a single hash.

For our timestamp network, we implement the proof-of-work by incrementing a nonce in the block until a value is found that gives the block's hash the required zero bits. Once the CPU effort has been expended to make it satisfy the proof-of-work, the block cannot be changed without redoing the work. As later blocks are chained after it, the work to change the block would include redoing all the blocks after it.



The proof-of-work also solves the problem of determining representation in majority decision making. If the majority were based on one-IP-address-one-vote, it could be subverted by anyone able to allocate many IPs. Proof-of-work is essentially one-CPU-one-vote. The majority decision is represented by the longest chain, which has the greatest proof-of-work effort invested in it. If a majority of CPU power is controlled by honest nodes, the honest chain will grow the fastest and outpace any competing chains. To modify a past block, an attacker would have to redo the proof-of-work of the block and all blocks after it and then catch up with and surpass the work of the honest nodes. We will show later that the probability of a slower attacker catching up diminishes exponentially as subsequent blocks are added.

To compensate for increasing hardware speed and varying interest in running nodes over time, the proof-of-work difficulty is determined by a moving average targeting an average number of blocks per hour. If they're generated too fast, the difficulty increases.

5. Network

The steps to run the network are as follows:

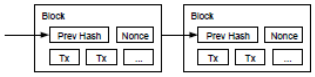
1) New transactions are broadcast to all nodes.
2) Each node collects new transactions into a block.
3) Each node works on finding a difficult proof-of-work for its block.
4) When a node finds a proof-of-work, it broadcasts the block to all nodes.
5) Nodes accept the block only if all transactions in it are valid and not already spent.
6) Nodes express their acceptance of the block by working on creating the next block in the chain, using the hash of the accepted block as the previous hash.

Nodes always consider the longest chain to be the correct one and will keep working on extending it. If two nodes broadcast different versions of the next block simultaneously, some nodes may receive one or the other first. In that case, they work on the first one they received, but save the other branch in case it becomes longer. The tie will be broken when the next proof-of-work is found and one branch becomes longer; the nodes that were working on the other branch will then switch to the longer one.

4. Proof-of-Work

To implement a distributed timestamp server on a peer-to-peer basis, we will need to use a proof-of-work system similar to Adam Back's Hashcash [6], rather than newspaper or Usenet posts. The proof-of-work involves scanning for a value that when hashed, such as with SHA-256, the hash begins with a number of zero bits. The average work required is exponential in the number of zero bits required and can be verified by executing a single hash.

For our timestamp network, we implement the proof-of-work by incrementing a nonce in the block until a value is found that gives the block's hash the required zero bits. Once the CPU effort has been expended to make it satisfy the proof-of-work, the block cannot be changed without redoing the work. As later blocks are chained after it, the work to change the block would include redoing all the blocks after it.



The proof-of-work also solves the problem of determining representation in majority decision making. If the majority were based on one-IP-address-one-vote, it could be subverted by anyone able to allocate many IPs. Proof-of-work is essentially one-CPU-one-vote. The majority decision is represented by the longest chain, which has the greatest proof-of-work effort invested in it. If a majority of CPU power is controlled by honest nodes, the honest chain will grow the fastest and outpace any competing chains. To modify a past block, an attacker would have to redo the proof-of-work of the block and all blocks after it and then catch up with and surpass the work of the honest nodes. We will show later that the probability of a slower attacker catching up diminishes exponentially as subsequent blocks are added.

To compensate for increasing hardware speed and varying interest in running nodes over time, the proof-of-work difficulty is determined by a moving average targeting an average number of blocks per hour. If they're generated too fast, the difficulty increases.

5. Network

The steps to run the network are as follows:

1) New transactions are broadcast to all nodes.
2) Each node collects new transactions into a block.
3) Each node works on finding a difficult proof-of-work for its block.
4) When a node finds a proof-of-work, it broadcasts the block to all nodes.
5) Nodes accept the block only if all transactions in it are valid and not already spent.
6) Nodes express their acceptance of the block by working on creating the next block in the chain, using the hash of the accepted block as the previous hash.

Nodes always consider the longest chain to be the correct one and will keep working on extending it. If two nodes broadcast different versions of the next block simultaneously, some nodes may receive one or the other first. In that case, they work on the first one they received, but save the other branch in case it becomes longer. The tie will be broken when the next proof-of-work is found and one branch becomes longer; the nodes that were working on the other branch will then switch to the longer one.

1307

Side 4:

New transaction broadcasts do not necessarily need to reach all nodes. As long as they reach many nodes, they will get into a block before long. Block broadcasts are also tolerant of dropped messages. If a node does not receive a block, it will request it when it receives the next block and realizes it missed one.

6. Incentive

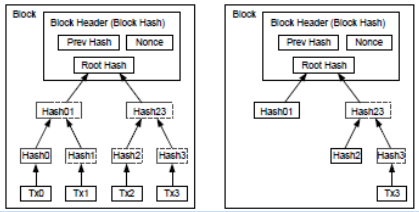
By convention, the first transaction in a block is a special transaction that starts a new coin owned by the creator of the block. This adds an incentive for nodes to support the network, and provides a way to initially distribute coins into circulation, since there is no central authority to issue them. The steady addition of a constant of amount of new coins is analogous to gold miners expending resources to add gold to circulation. In our case, it is CPU time and electricity that is expended.

The incentive can also be funded with transaction fees. If the output value of a transaction is less than its input value, the difference is a transaction fee that is added to the incentive value of the block containing the transaction. Once a predetermined number of coins have entered circulation, the incentive can transition entirely to transaction fees and be completely inflation free.

The incentive may help encourage nodes to stay honest. If a greedy attacker is able to assemble more CPU power than all the honest nodes, he would have to choose between using it to defraud people by stealing back his payments, or using it to generate new coins. He ought to find it more profitable to play by the rules, such rules that favour him with more new coins than everyone else combined, than to undermine the system and the validity of his own wealth.

7. Reclaiming Disk Space

Once the latest transaction in a coin is buried under enough blocks, the spent transactions before it can be discarded to save disk space. To facilitate this without breaking the block's hash, transactions are hashed in a Merkle Tree [7][2][5], with only the root included in the block's hash. Old blocks can then be compacted by stubbing off branches of the tree. The interior hashes do not need to be stored.



A block header with no transactions would be about 80 bytes. If we suppose blocks are generated every 10 minutes, 80 bytes * 6 * 24 * 365 = 4.2MB per year. With computer systems typically selling with 2GB of RAM as of 2008, and Moore's Law predicting current growth of 1.2GB per year, storage should not be a problem even if the block headers must be kept in memory.

New transaction broadcasts do not necessarily need to reach all nodes. As long as they reach many nodes, they will get into a block before long. Block broadcasts are also tolerant of dropped messages. If a node does not receive a block, it will request it when it receives the next block and realizes it missed one.

6. Incentive

By convention, the first transaction in a block is a special transaction that starts a new coin owned by the creator of the block. This adds an incentive for nodes to support the network, and provides a way to initially distribute coins into circulation, since there is no central authority to issue them. The steady addition of a constant of amount of new coins is analogous to gold miners expending resources to add gold to circulation. In our case, it is CPU time and electricity that is expended.

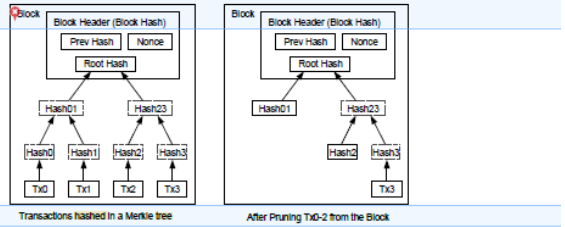
The incentive can also be funded with transaction fees. If the output value of a transaction is less than its input value, the difference is a transaction fee that is added to the incentive value of the block containing the transaction. Once a predetermined number of coins have entered circulation, the incentive can transition entirely to transaction fees and be completely inflation free.

The incentive may help encourage nodes to stay honest. If a greedy attacker is able to assemble more CPU power than all the honest nodes, he would have to choose between using it to defraud people by stealing back his payments, or using it to generate new coins. He ought to find it more profitable to play by the rules, such rules that favour him with more new coins than everyone else combined, than to undermine the system and the validity of his own wealth.

To compensate for increasing hardware speed and varying interest in running nodes over time, the proof-of-work difficulty is determined by a moving average targeting an average number of blocks per hour. If they're generated too fast, the difficulty increases.

7. Reclaiming Disk Space

Once the latest transaction in a coin is buried under enough blocks, the spent transactions before it can be discarded to save disk space. To facilitate this without breaking the block's hash, transactions are hashed in a Merkle Tree [7][2][5], with only the root included in the block's hash. Old blocks can then be compacted by stubbing off branches of the tree. The interior hashes do not need to be stored.



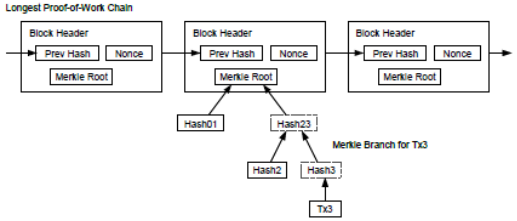
A block header with no transactions would be about 80 bytes. If we suppose blocks are generated every 10 minutes, 80 bytes * 6 * 24 * 365 = 4.2MB per year. With computer systems typically selling with 2GB of RAM as of 2008, and Moore's Law predicting current growth of 1.2GB per year, storage should not be a problem even if the block headers must be kept in memory.

1309

Side 5:

8. Simplified Payment Verification

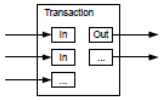
It is possible to verify payments without running a full network node. A user only needs to keep a copy of the block headers of the longest proof-of-work chain, which he can get by querying network nodes until he's convinced he has the longest chain, and obtain the Merkle branch linking the transaction to the block it's timestamped in. He can't check the transaction for himself, but by linking it to a place in the chain, he can see that a network node has accepted it, and blocks added after it further confirm the network has accepted it.



As such, the verification is reliable as long as honest nodes control the network, but is more vulnerable if the network is overpowered by an attacker. While network nodes can verify transactions for themselves, the simplified method can be fooled by an attacker's fabricated transactions for as long as the attacker can continue to overpower the network. One strategy to protect against this would be to accept alerts from network nodes when they detect an invalid block, prompting the user's software to download the full block and alerted transactions to confirm the inconsistency. Businesses that receive frequent payments will probably still want to run their own nodes for more independent security and quicker verification.

9. Combining and Splitting Value

Although it would be possible to handle coins individually, it would be unwieldy to make a separate transaction for every cent in a transfer. To allow value to be split and combined, transactions contain multiple inputs and outputs. Normally there will be either a single input from a larger previous transaction or multiple inputs combining smaller amounts, and at most two outputs: one for the payment, and one returning the change, if any, back to the sender.



It should be noted that fan-out, where a transaction depends on several transactions, and those transactions depend on many more, is not a problem here. There is never the need to extract a complete standalone copy of a transaction's history.

5

8. Simplified Payment Verification

It is possible to verify payments without running a full network node. A user only needs to keep a copy of the block headers of the longest proof-of-work chain, which he can get by querying network nodes until he's convinced he has the longest chain, and obtain the Merkle branch linking the transaction to the block it's timestamped in. He can't check the transaction for himself, but by linking it to a place in the chain, he can see that a network node has accepted it, and blocks added after it further confirm the network has accepted it.

Longest Proof-of-Work Chain
Block Header
Prev Hash
Nonce
Merkle Root
Block Header
Prev Hash
Nonce
Merkle Root
Block Header
Prev Hash
Nonce
Merkle Root
Hash01
Hash23
Merkle Branch for Tx3
Hash2
Hash3
Tx3

As such, the verification is reliable as long as honest nodes control the network, but is more vulnerable if the network is overpowered by an attacker. While network nodes can verify transactions for themselves and are only vulnerable to reversal, the simplified method can be fooled by an attacker's fabricated transactions for as long as the attacker can continue to overpower the network. One strategy to protect against this would be to accept alerts from network nodes when they detect an invalid block, prompting the user's software to download the full block and alerted transactions to confirm the inconsistency. Businesses that receive frequent payments will probably still want to run their own nodes for more independent security and quicker verification.

9. Combining and Splitting Value

Although it would be possible to handle coins individually, it would be unwieldy to make a separate transaction for every cent in a transfer. To allow value to be split and combined, transactions contain multiple inputs and outputs. Normally there will be either a single input from a larger previous transaction or multiple inputs combining smaller amounts, and at most two outputs: one for the payment, and one returning the change, if any, back to the sender.

Transaction
In
Out
In
...
...

It should be noted that fan-out, where a transaction depends on several transactions, and those transactions depend on many more, is not a problem here. There is never the need to extract a complete standalone copy of a transaction's history.

5

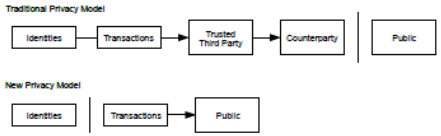
1309

1310

Side 6:

10. Privacy

The traditional banking model achieves a level of privacy by limiting access to information to the parties involved and the trusted third party. The necessity to announce all transactions publicly precludes this method, but privacy can still be maintained by breaking the flow of information in another place: by keeping public keys anonymous. The public can see that someone is sending an amount to someone else, but without information linking the transaction to anyone. This is similar to the level of information released by stock exchanges, where the time and size of individual trades, the "tape", is made public, but without telling who the parties were.



As an additional firewall, a new key pair should be used for each transaction to keep them from being linked to a common owner. Some linking is still unavoidable with multi-input transactions, which necessarily reveal that their inputs were owned by the same owner. The risk is that if the owner of a key is revealed, linking could reveal other transactions that belonged to the same owner.

11. Calculations

We consider the scenario of an attacker trying to generate an alternate chain faster than the honest chain. Even if this is accomplished, it does not throw the system open to arbitrary changes, such as creating value out of thin air or taking money that never belonged to the attacker. Nodes are not going to accept an invalid transaction as payment, and honest nodes will never accept a block containing them. An attacker can only try to change one of his own transactions to take back money he recently spent.

The race between the honest chain and an attacker chain can be characterized as a Binomial Random Walk. The success event is the honest chain being extended by one block, increasing its lead by +1, and the failure event is the attacker's chain being extended by one block, reducing the gap by -1.

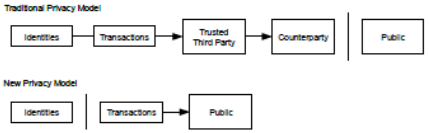
The probability of an attacker catching up from a given deficit is analogous to a Gambler's Ruin problem. Suppose a gambler with unlimited credit starts at a deficit and plays potentially an infinite number of trials to try to reach breakeven. We can calculate the probability he ever reaches breakeven, or that an attacker ever catches up with the honest chain, as follows [8]:

p = probability an honest node finds the next block
q = probability the attacker finds the next block
q_z = probability the attacker will ever catch up from z blocks behind

$$q_z = \begin{cases} 1 & \text{if } p \le q \\ (q/p)^z & \text{if } p > q \end{cases}$$

10. Privacy

The traditional banking model achieves a level of privacy by limiting access to information to the parties involved and the trusted third party. The necessity to announce all transactions publicly precludes this method, but privacy can still be maintained by breaking the flow of information in another place: by keeping public keys anonymous. The public can see that someone is sending an amount to someone else, but without information linking the transaction to anyone. This is similar to the level of information released by stock exchanges, where the time and size of individual trades, the "tape", is made public, but without telling who the parties were.



As an additional firewall, a new key pair should be used for each transaction to keep them from being linked to a common owner. Some linking is still unavoidable with multi-input transactions, which necessarily reveal that their inputs were owned by the same owner. The risk is that if the owner of a key is revealed, linking could reveal other transactions that belonged to the same owner.

11. Calculations

We consider the scenario of an attacker trying to generate an alternate chain faster than the honest chain. Even if this is accomplished, it does not throw the system open to arbitrary changes, such as creating value out of thin air or taking money that never belonged to the attacker. Nodes are not going to accept an invalid transaction as payment, and honest nodes will never accept a block containing them. An attacker can only try to change one of his own transactions to take back money he recently spent.

The race between the honest chain and an attacker chain can be characterized as a Binomial Random Walk. The success event is the honest chain being extended by one block, increasing its lead by +1, and the failure event is the attacker's chain being extended by one block, reducing the gap by -1.

The probability of an attacker catching up from a given deficit is analogous to a Gambler's Ruin problem. Suppose a gambler with unlimited credit starts at a deficit and plays potentially an infinite number of trials to try to reach breakeven. We can calculate the probability he ever reaches breakeven, or that an attacker ever catches up with the honest chain, as follows [8]:

p = probability an honest node finds the next block
q = probability the attacker finds the next block
q_z = probability the attacker will ever catch up from z blocks behind

$$q_z = \begin{cases} 1 & \text{if } p \le q \\ (q/p)^z & \text{if } p > q \end{cases}$$

6

1310

1311

Side 7:

Given our assumption that $p > q$, the probability drops exponentially as the number of blocks the attacker has to catch up with increases. With the odds against him, if he doesn't make a lucky lunge forward early on, his chances become vanishingly small as he falls further behind.

We now consider how long the recipient of a new transaction needs to wait before being sufficiently certain the sender can't change the transaction. We assume the sender is an attacker who wants to make the recipient believe he paid him for a while, then switch it to pay back to himself after some time has passed. The receiver will be alerted when that happens, but the sender hopes it will be too late.

The receiver generates a new key pair and gives the public key to the sender shortly before signing. This prevents the sender from preparing a chain of blocks ahead of time by working on it continuously until he is lucky enough to get far enough ahead, then executing the transaction at that moment. Once the transaction is sent, the dishonest sender starts working in secret on a parallel chain containing an alternate version of his transaction.

The recipient waits until the transaction has been added to a block and z blocks have been linked after it. He doesn't know the exact amount of progress the attacker has made, but assuming the honest blocks took the average expected time per block, the attacker's potential progress will be a Poisson distribution with expected value:

$$\lambda = z\frac{q}{p}$$

To get the probability the attacker could still catch up now, we multiply the Poisson density for each amount of progress he could have made by the probability he could catch up from that point:

$$\sum_{k=0}^{\infty} \frac{\lambda^k e^{-\lambda}}{k!} \cdot \begin{cases} (q/p)^{(z-k)} & \text{if } k \le z \\ 1 & \text{if } k > z \end{cases}$$

Rearranging to avoid summing the infinite tail of the distribution...

$$1 - \sum_{k=0}^{z} \frac{\lambda^k e^{-\lambda}}{k!} \left(1 - (q/p)^{(z-k)}\right)$$

Converting to C code...

```
#include <math.h>
double AttackerSuccessProbability(double q, int z)
{
    double p = 1.0 - q;
    double lambda = z * (q / p);
    double sum = 1.0;
    int i, k;
    for (k = 0; k <= z; k++)
    {
        double poisson = exp(-lambda);
        for (i = 1; i <= k; i++)
            poisson *= lambda / i;
        sum -= poisson * (1 - pow(q / p, z - k));
    }
    return sum;
}
```


Given our assumption that $p > q$, the probability drops exponentially as the number of blocks the attacker has to catch up with increases. With the odds against him, if he doesn't make a lucky lunge forward early on, his chances become vanishingly small as he falls further behind.

We now consider how long the recipient of a new transaction needs to wait before being sufficiently certain the sender can't change the transaction. We assume the sender is an attacker who wants to make the recipient believe he paid him for a while, then switch it to pay back to himself after some time has passed. The receiver will be alerted when that happens, but the sender hopes it will be too late.

The receiver generates a new key pair and gives the public key to the sender shortly before signing. This prevents the sender from preparing a chain of blocks ahead of time by working on it continuously until he is lucky enough to get far enough ahead, then executing the transaction at that moment. Once the transaction is sent, the dishonest sender starts working in secret on a parallel chain containing an alternate version of his transaction.

The recipient waits until the transaction has been added to a block and z blocks have been linked after it. He doesn't know the exact amount of progress the attacker has made, but assuming the honest blocks took the average expected time per block, the attacker's potential progress will be a Poisson distribution with expected value:

$$\lambda = z\frac{q}{p}$$

To get the probability the attacker could still catch up now, we multiply the Poisson density for each amount of progress he could have made by the probability he could catch up from that point:

$$\sum_{k=0}^{\infty} \frac{\lambda^k e^{-\lambda}}{k!} \cdot \begin{cases} (q/p)^{(z-k)} & \text{if } k \le z \\ 1 & \text{if } k > z \end{cases}$$

Rearranging to avoid summing the infinite tail of the distribution...

$$1 - \sum_{k=0}^{z} \frac{\lambda^k e^{-\lambda}}{k!} \left(1 - (q/p)^{(z-k)}\right)$$

Converting to C code...

```
#include <math.h>
double AttackerSuccessProbability(double q, int z)
{
    double p = 1.0 - q;
    double lambda = z * (q / p);
    double sum = 1.0;
    int i, k;
    for (k = 0; k <= z; k++)
    {
        double poisson = exp(-lambda);
        for (i = 1; i <= k; i++)
            poisson *= lambda / i;
        sum -= poisson * (1 - pow(q / p, z - k));
    }
    return sum;
}
```


1311

1312

Side 8:

Running some results, we can see the probability drop off exponentially with z.

```
q=0.1
z=0     P=1.0000000
z=1     P=0.2045873
z=2     P=0.0509779
z=3     P=0.0131722
z=4     P=0.0034552
z=5     P=0.0009137
z=6     P=0.0002428
z=7     P=0.0000647
z=8     P=0.0000173
z=9     P=0.0000046
z=10    P=0.0000012

q=0.3
z=0     P=1.0000000
z=5     P=0.1773523
z=10    P=0.0416605
z=15    P=0.0101008
z=20    P=0.0024804
z=25    P=0.0006132
z=30    P=0.0001522
z=35    P=0.0000379
z=40    P=0.0000095
z=45    P=0.0000024
z=50    P=0.0000006
```

Solving for P less than 0.1%...

```
P < 0.001
q=0.10   z=5
q=0.15   z=8
q=0.20   z=11
q=0.25   z=15
q=0.30   z=24
q=0.35   z=41
q=0.40   z=89
q=0.45   z=340
```

12. Conclusion

We have proposed a system for electronic transactions without relying on trust. We started with the usual framework of coins made from digital signatures, which provides strong control of ownership, but is incomplete without a way to prevent double-spending. To solve this, we proposed a peer-to-peer network using proof-of-work to record a public history of transactions that quickly becomes computationally impractical for an attacker to change if honest nodes control a majority of CPU power. The network is robust in its unstructured simplicity. Nodes work all at once with little coordination. They do not need to be identified, since messages are not routed to any particular place and only need to be delivered on a best effort basis. Nodes can leave and rejoin the network at will, accepting the proof-of-work chain as proof of what happened while they were gone. They vote with their CPU power, expressing their acceptance of valid blocks by working on extending them and rejecting invalid blocks by refusing to work on them. Any needed rules and incentives can be enforced with this consensus mechanism.

Running some results, we can see the probability drop off exponentially with z.

```
q=0.1
z=0     P=1.0000000
z=1     P=0.2045873
z=2     P=0.0509779
z=3     P=0.0131722
z=4     P=0.0034552
z=5     P=0.0009137
z=6     P=0.0002428
z=7     P=0.0000647
z=8     P=0.0000173
z=9     P=0.0000046
z=10    P=0.0000012

q=0.3
z=0     P=1.0000000
z=5     P=0.1773523
z=10    P=0.0416605
z=15    P=0.0101008
z=20    P=0.0024804
z=25    P=0.0006132
z=30    P=0.0001522
z=35    P=0.0000379
z=40    P=0.0000095
z=45    P=0.0000024
z=50    P=0.0000006
```

Solving for P less than 0.1%...

```
P < 0.001
q=0.10   z=5
q=0.15   z=8
q=0.20   z=11
q=0.25   z=15
q=0.30   z=24
q=0.35   z=41
q=0.40   z=89
q=0.45   z=340
```

12. Conclusion

We have proposed a system for electronic transactions without relying on trust. We started with the usual framework of coins made from digital signatures, which provides strong control of ownership, but is incomplete without a way to prevent double-spending. To solve this, we proposed a peer-to-peer network using proof-of-work to record a public history of transactions that quickly becomes computationally impractical for an attacker to change if honest nodes control a majority of CPU power. The network is robust in its unstructured simplicity. Nodes work all at once with little coordination. They do not need to be identified, since messages are not routed to any particular place and only need to be delivered on a best effort basis. Nodes can leave and rejoin the network at will, accepting the proof-of-work chain as proof of what happened while they were gone. They vote with their CPU power, expressing their acceptance of valid blocks by working on extending them and rejecting invalid blocks by refusing to work on them. Any needed rules and incentives can be enforced with this consensus mechanism.

1312

1313

Side 9:

References

[1] W. Dai, "b-money," http://www.weidai.com/bmoney.txt, 1998.

[2] H. Massias, X.S. Avila, and J.-J. Quisquater, "Design of a secure timestamping service with minimal trust requirements," In *20th Symposium on Information Theory in the Benelux*, May 1999.

[3] S. Haber, W.S. Stornetta, "How to time-stamp a digital document," In *Journal of Cryptology*, vol 3, no 2, pages 99-111, 1991.

[4] D. Bayer, S. Haber, W.S. Stornetta, "Improving the efficiency and reliability of digital time-stamping," In *Sequences II: Methods in Communication, Security and Computer Science*, pages 329-334, 1993.

[5] S. Haber, W.S. Stornetta, "Secure names for bit-strings," In *Proceedings of the 4th ACM Conference on Computer and Communications Security*, pages 28-35, April 1997.

[6] A. Back, "Hashcash - a denial of service counter-measure," http://www.hashcash.org/papers/hashcash.pdf, 2002.

[7] R.C. Merkle, "Protocols for public key cryptosystems," In *Proc. 1980 Symposium on Security and Privacy*, IEEE Computer Society, pages 122-133, April 1980.

[8] W. Feller, "An introduction to probability theory and its applications," 1957.

References

[1] W. Dai, "b-money," http://www.weidai.com/bmoney.txt, 1998.

[2] H. Massias, X.S. Avila, and J.-J. Quisquater, "Design of a secure timestamping service with minimal trust requirements," In *20th Symposium on Information Theory in the Benelux*, May 1999.

[3] S. Haber, W.S. Stornetta, "How to time-stamp a digital document," In *Journal of Cryptology*, vol 3, no 2, pages 99-111, 1991.

[4] D. Bayer, S. Haber, W.S. Stornetta, "Improving the efficiency and reliability of digital time-stamping," In *Sequences II: Methods in Communication, Security and Computer Science*, pages 329-334, 1993.

[5] S. Haber, W.S. Stornetta, "Secure names for bit-strings," In *Proceedings of the 4th ACM Conference on Computer and Communications Security*, pages 28-35, April 1997.

[6] A. Back, "Hashcash - a denial of service counter-measure," http://www.hashcash.org/papers/hashcash.pdf, 2002.

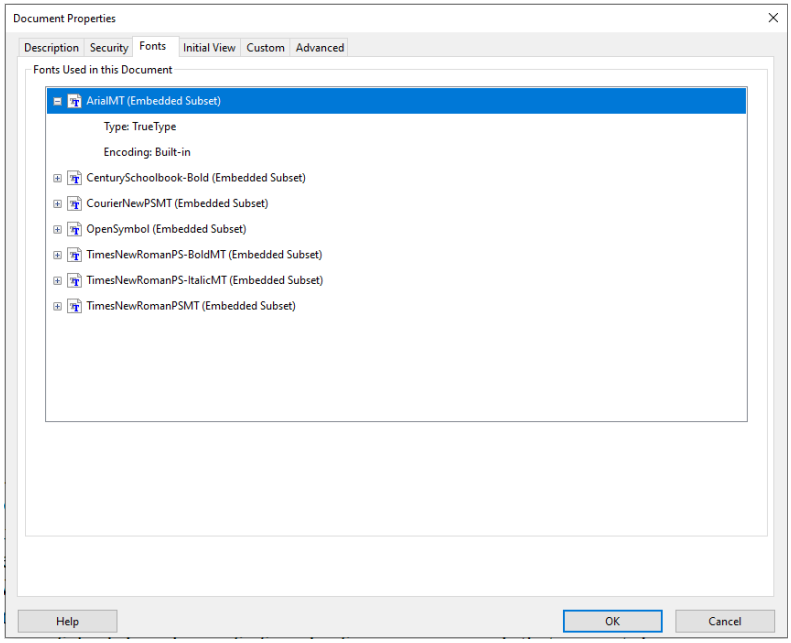
[7] R.C. Merkle, "Protocols for public key cryptosystems," In *Proc. 1980 Symposium on Security and Privacy*, IEEE Computer Society, pages 122-133, April 1980.

[8] W. Feller, "An introduction to probability theory and its applications," 1957.

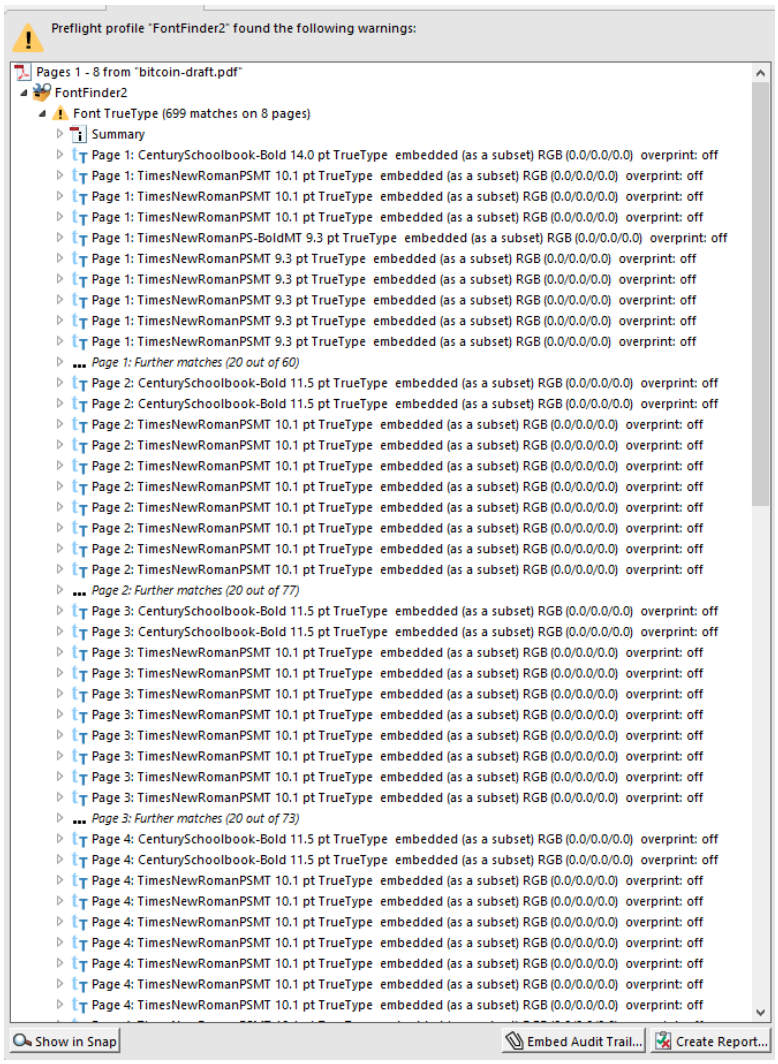
1313

4.17 Vedlegg 17: Innholdsanalyse «bitcoin-draft.pdf»

4.17.1 Fontoversikt



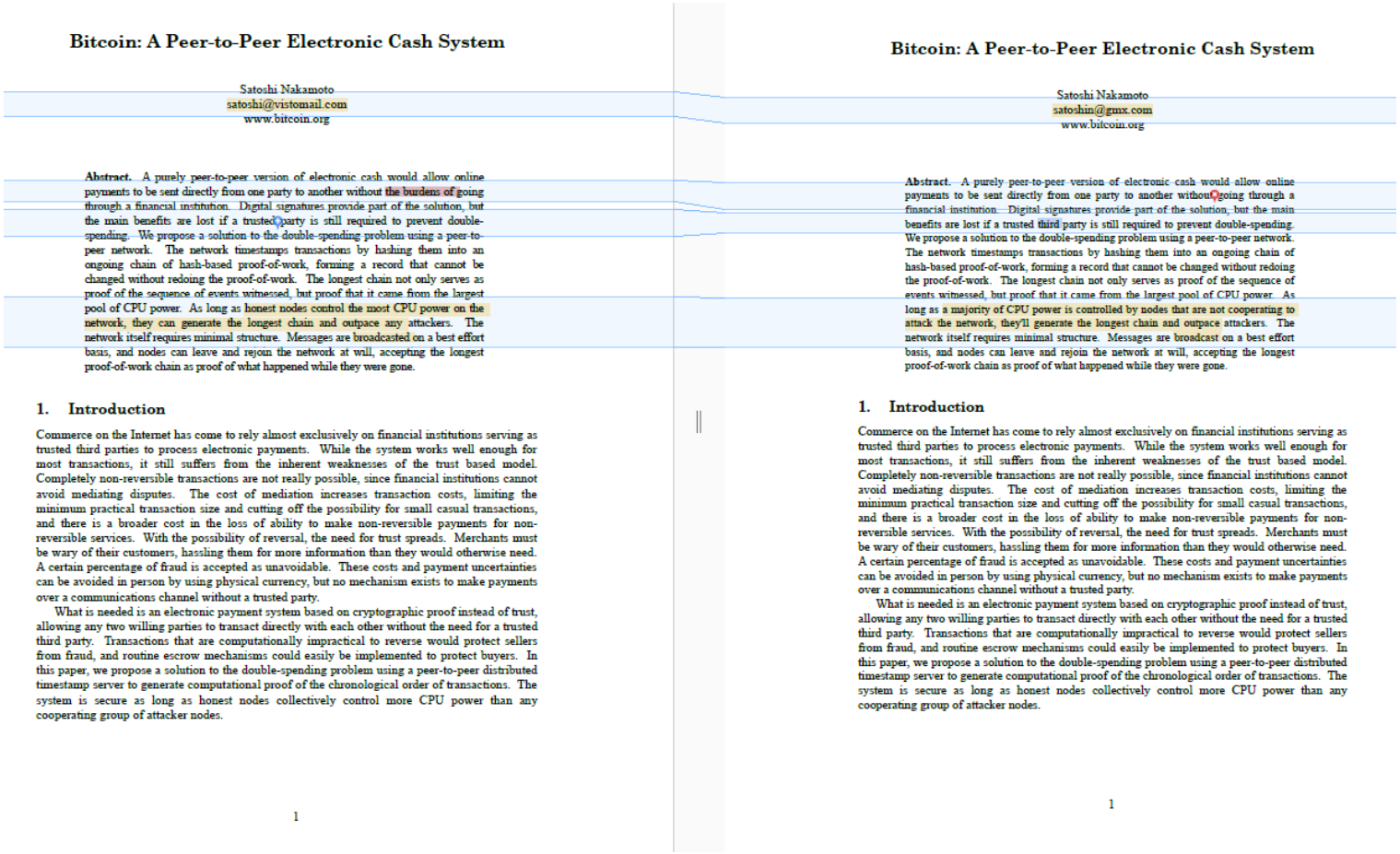
4.17.2 Font-type



4.17.3 Tekstforskjeller fra «bitcoin-draft.pdf» vs. «bitcoin.pdf»

All «endret» tekst er uthevet i gult, all tillegg av tekst er uthevet i blått og andre endringer er markert med et hvitt kryss på rød bakgrunn.

Side 1:



Bitcoin: A Peer-to-Peer Electronic Cash System

Satoshi Nakamoto
satoshi@vistomail.com
www.bitcoin.org

Abstract. A purely peer-to-peer version of electronic cash would allow online payments to be sent directly from one party to another without the burdens of going through a financial institution. Digital signatures provide part of the solution, but the main benefits are lost if a trusted party is still required to prevent double-spending. We propose a solution to the double-spending problem using a peer-to-peer network. The network timestamps transactions by hashing them into an ongoing chain of hash-based proof-of-work, forming a record that cannot be changed without redoing the proof-of-work. The longest chain not only serves as proof of the sequence of events witnessed, but proof that it came from the largest pool of CPU power. As long as honest nodes control the most CPU power on the network, they can generate the longest chain and outpace any attackers. The network itself requires minimal structure. Messages are broadcasted on a best effort basis, and nodes can leave and rejoin the network at will, accepting the longest proof-of-work chain as proof of what happened while they were gone.

1. Introduction

Commerce on the Internet has come to rely almost exclusively on financial institutions serving as trusted third parties to process electronic payments. While the system works well enough for most transactions, it still suffers from the inherent weaknesses of the trust based model. Completely non-reversible transactions are not really possible, since financial institutions cannot avoid mediating disputes. The cost of mediation increases transaction costs, limiting the minimum practical transaction size and cutting off the possibility for small casual transactions, and there is a broader cost in the loss of ability to make non-reversible payments for non-reversible services. With the possibility of reversal, the need for trust spreads. Merchants must be wary of their customers, hassling them for more information than they would otherwise need. A certain percentage of fraud is accepted as unavoidable. These costs and payment uncertainties can be avoided in person by using physical currency, but no mechanism exists to make payments over a communications channel without a trusted party.

What is needed is an electronic payment system based on cryptographic proof instead of trust, allowing any two willing parties to transact directly with each other without the need for a trusted third party. Transactions that are computationally impractical to reverse would protect sellers from fraud, and routine escrow mechanisms could easily be implemented to protect buyers. In this paper, we propose a solution to the double-spending problem using a peer-to-peer distributed timestamp server to generate computational proof of the chronological order of transactions. The system is secure as long as honest nodes collectively control more CPU power than any cooperating group of attacker nodes.

1

Bitcoin: A Peer-to-Peer Electronic Cash System

Satoshi Nakamoto
satoshin@gmx.com
www.bitcoin.org

Abstract. A purely peer-to-peer version of electronic cash would allow online payments to be sent directly from one party to another without going through a financial institution. Digital signatures provide part of the solution, but the main benefits are lost if a trusted third party is still required to prevent double-spending. We propose a solution to the double-spending problem using a peer-to-peer network. The network timestamps transactions by hashing them into an ongoing chain of hash-based proof-of-work, forming a record that cannot be changed without redoing the proof-of-work. The longest chain not only serves as proof of the sequence of events witnessed, but proof that it came from the largest pool of CPU power. As long as a majority of CPU power is controlled by nodes that are not cooperating to attack the network, they'll generate the longest chain and outpace attackers. The network itself requires minimal structure. Messages are broadcast on a best effort basis, and nodes can leave and rejoin the network at will, accepting the longest proof-of-work chain as proof of what happened while they were gone.

1. Introduction

Commerce on the Internet has come to rely almost exclusively on financial institutions serving as trusted third parties to process electronic payments. While the system works well enough for most transactions, it still suffers from the inherent weaknesses of the trust based model. Completely non-reversible transactions are not really possible, since financial institutions cannot avoid mediating disputes. The cost of mediation increases transaction costs, limiting the minimum practical transaction size and cutting off the possibility for small casual transactions, and there is a broader cost in the loss of ability to make non-reversible payments for non-reversible services. With the possibility of reversal, the need for trust spreads. Merchants must be wary of their customers, hassling them for more information than they would otherwise need. A certain percentage of fraud is accepted as unavoidable. These costs and payment uncertainties can be avoided in person by using physical currency, but no mechanism exists to make payments over a communications channel without a trusted party.

What is needed is an electronic payment system based on cryptographic proof instead of trust, allowing any two willing parties to transact directly with each other without the need for a trusted third party. Transactions that are computationally impractical to reverse would protect sellers from fraud, and routine escrow mechanisms could easily be implemented to protect buyers. In this paper, we propose a solution to the double-spending problem using a peer-to-peer distributed timestamp server to generate computational proof of the chronological order of transactions. The system is secure as long as honest nodes collectively control more CPU power than any cooperating group of attacker nodes.

1

1317

Side 2:

Ingen forskjeller

1317

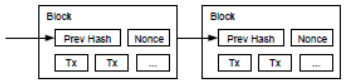
1318

Side 3:

4. Proof-of-Work

To implement a distributed timestamp server on a peer-to-peer basis, we will need to use a proof-of-work system similar to Adam Back's Hashcash [6], rather than newspaper or Usenet posts. The proof-of-work involves scanning for a value that when hashed, such as with SHA-256, the hash begins with a number of zero bits. The average work required is exponential in the number of zero bits required and can be verified by executing a single hash.

For our timestamp network, we implement the proof-of-work by incrementing a nonce in the block until a value is found that gives the block's hash the required zero bits. Once the CPU effort has been expended to make it satisfy the proof-of-work, the block cannot be changed without redoing the work. As later blocks are chained after it, the work to change the block would include redoing all the blocks after it.



The proof-of-work also solves the problem of determining representation in majority decision making. If the majority were based on one-IP-address-one-vote, it could be subverted by anyone able to allocate many IPs. Proof-of-work is essentially one-CPU-one-vote. The majority decision is represented by the longest chain, which has the greatest proof-of-work effort invested in it. If a majority of CPU power is controlled by honest nodes, the honest chain will grow the fastest and outpace any competing chains. To modify a past block, an attacker would have to redo the proof-of-work of the block and all blocks after it and then catch up with and surpass the work of the honest nodes. We will show later that the probability of a slower attacker catching up diminishes exponentially as subsequent blocks are added.

5. Network

The steps to run the network are as follows:

1) New transactions are broadcasted to all nodes.
2) Each node collects new transactions into a block.
3) Each node works on finding a difficult proof-of-work for its block.
4) When a node finds a proof-of-work, it broadcasts the block to all nodes.
5) Nodes accept the block only if all transactions in it are valid and not already spent.
6) Nodes express their acceptance of the block by working on creating the next block in the chain, using the hash of the accepted block as the previous hash.

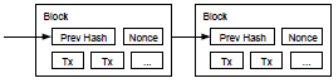
Nodes always consider the longest chain to be the correct one and will keep working on extending it. If two nodes broadcast different versions of the next block simultaneously, some nodes may receive one or the other first. In that case, they work on the first one they received, but save the other branch in case it becomes longer. The tie will be broken when the next proof-of-work is found and one branch becomes longer; the nodes that were working on the other branch will then switch to the longer one.

New transaction broadcasts do not necessarily need to reach all nodes. As long as they reach many nodes, they will get into a block before long. Block broadcasts are also tolerant of dropped messages. If a node does not receive a block, it will request it when it receives the next block and realizes it missed one.

4. Proof-of-Work

To implement a distributed timestamp server on a peer-to-peer basis, we will need to use a proof-of-work system similar to Adam Back's Hashcash [6], rather than newspaper or Usenet posts. The proof-of-work involves scanning for a value that when hashed, such as with SHA-256, the hash begins with a number of zero bits. The average work required is exponential in the number of zero bits required and can be verified by executing a single hash.

For our timestamp network, we implement the proof-of-work by incrementing a nonce in the block until a value is found that gives the block's hash the required zero bits. Once the CPU effort has been expended to make it satisfy the proof-of-work, the block cannot be changed without redoing the work. As later blocks are chained after it, the work to change the block would include redoing all the blocks after it.



The proof-of-work also solves the problem of determining representation in majority decision making. If the majority were based on one-IP-address-one-vote, it could be subverted by anyone able to allocate many IPs. Proof-of-work is essentially one-CPU-one-vote. The majority decision is represented by the longest chain, which has the greatest proof-of-work effort invested in it. If a majority of CPU power is controlled by honest nodes, the honest chain will grow the fastest and outpace any competing chains. To modify a past block, an attacker would have to redo the proof-of-work of the block and all blocks after it and then catch up with and surpass the work of the honest nodes. We will show later that the probability of a slower attacker catching up diminishes exponentially as subsequent blocks are added.

To compensate for increasing hardware speed and varying interest in running nodes over time, the proof-of-work difficulty is determined by a moving average targeting an average number of blocks per hour. If they're generated too fast, the difficulty increases.

5. Network

The steps to run the network are as follows:

1) New transactions are broadcast to all nodes.
2) Each node collects new transactions into a block.
3) Each node works on finding a difficult proof-of-work for its block.
4) When a node finds a proof-of-work, it broadcasts the block to all nodes.
5) Nodes accept the block only if all transactions in it are valid and not already spent.
6) Nodes express their acceptance of the block by working on creating the next block in the chain, using the hash of the accepted block as the previous hash.

Nodes always consider the longest chain to be the correct one and will keep working on extending it. If two nodes broadcast different versions of the next block simultaneously, some nodes may receive one or the other first. In that case, they work on the first one they received, but save the other branch in case it becomes longer. The tie will be broken when the next proof-of-work is found and one branch becomes longer; the nodes that were working on the other branch will then switch to the longer one.

1318

1319

Side 4:

6. Incentive

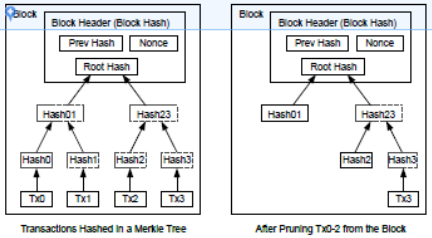
By convention, the first transaction in a block is a special transaction that starts a new coin owned by the creator of the block. This adds an incentive for nodes to support the network, and provides a way to initially distribute coins into circulation, since there is no central authority to issue them. The steady addition of a constant of amount of new coins is analogous to gold miners expending resources to add gold to circulation. In our case, it is CPU time and electricity that is expended.

The incentive may also help encourage nodes to stay honest. If a greedy attacker is able to assemble more CPU power than all the honest nodes, he would have to choose between using it to defraud people by stealing back his payments, or using it to generate new coins. He ought to find it more profitable to play by the rules, such rules that favour him with more new coins than everyone else combined, than to undermine the system and the validity of his own wealth.

To compensate for increasing hardware speed and varying interest in running nodes over time, the proof-of-work difficulty is determined by a moving average targeting an average number of blocks per hour. If they're generated too fast, the difficulty increases.

7. Reclaiming Disk Space

Once the latest transaction in a coin is buried under enough blocks, the spent transactions before it can be discarded to save disk space. To facilitate this without breaking the block's hash, transactions are hashed in a Merkle Tree [7][2][5], with only the root included in the block's hash. Old blocks can then be compacted by stubbing off branches of the tree. The interior hashes do not need to be stored.



A block header with no transactions would be about 80 bytes. If we suppose blocks are generated every 10 minutes, 80 bytes * 6 * 24 * 365 = 4.2MB per year. With computer systems typically selling with 2GB of RAM as of 2008, and Moore's Law predicting current growth of 1.2GB per year, storage should not be a problem even if the block headers must be kept in memory.

8. Simplified Payment Verification

It is possible to verify payments without running a full network node. A user only needs to keep a copy of the block headers of the longest proof-of-work chain, which he can get by querying network nodes until he's convinced he has the longest chain, and obtain the Merkle branch linking the transaction to the block it's timestamped in. He can't check the transaction for

New transaction broadcasts do not necessarily need to reach all nodes. As long as they reach many nodes, they will get into a block before long. Block broadcasts are also tolerant of dropped messages. If a node does not receive a block, it will request it when it receives the next block and realizes it missed one.

6. Incentive

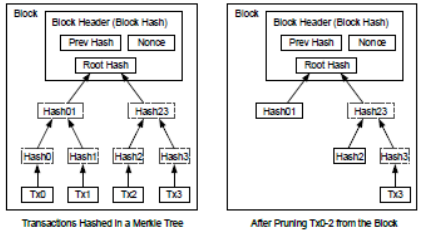
By convention, the first transaction in a block is a special transaction that starts a new coin owned by the creator of the block. This adds an incentive for nodes to support the network, and provides a way to initially distribute coins into circulation, since there is no central authority to issue them. The steady addition of a constant of amount of new coins is analogous to gold miners expending resources to add gold to circulation. In our case, it is CPU time and electricity that is expended.

The incentive can also be funded with transaction fees. If the output value of a transaction is less than its input value, the difference is a transaction fee that is added to the incentive value of the block containing the transaction. Once a predetermined number of coins have entered circulation, the incentive can transition entirely to transaction fees and be completely inflation free.

The incentive may help encourage nodes to stay honest. If a greedy attacker is able to assemble more CPU power than all the honest nodes, he would have to choose between using it to defraud people by stealing back his payments, or using it to generate new coins. He ought to find it more profitable to play by the rules, such rules that favour him with more new coins than everyone else combined, than to undermine the system and the validity of his own wealth.

7. Reclaiming Disk Space

Once the latest transaction in a coin is buried under enough blocks, the spent transactions before it can be discarded to save disk space. To facilitate this without breaking the block's hash, transactions are hashed in a Merkle Tree [7][2][5], with only the root included in the block's hash. Old blocks can then be compacted by stubbing off branches of the tree. The interior hashes do not need to be stored.



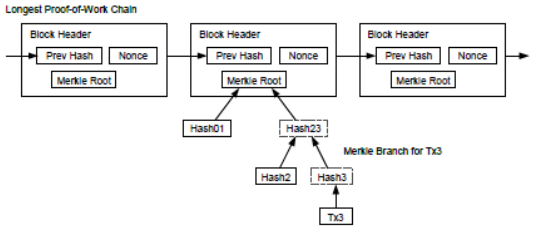
A block header with no transactions would be about 80 bytes. If we suppose blocks are generated every 10 minutes, 80 bytes * 6 * 24 * 365 = 4.2MB per year. With computer systems typically selling with 2GB of RAM as of 2008, and Moore's Law predicting current growth of 1.2GB per year, storage should not be a problem even if the block headers must be kept in memory.

1319

1320

Side 5:

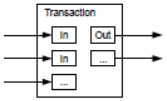
himself, but by linking it to a place in the chain, he can see that a network node has accepted it, and blocks added after it further confirm the network has accepted it.



As such, the verification is reliable as long as honest nodes control the network, but is more vulnerable if the network is overpowered by an attacker. While network nodes can verify transactions for themselves and are only vulnerable to reversal, the simplified method can be fooled by an attacker's fabricated transactions for as long as the attacker can continue to overpower the network. One strategy to protect against this would be to accept alerts from network nodes when they detect an invalid block, prompting the user's software to download the full block and reported transactions to confirm the inconsistency. Businesses that receive frequent payments will probably still want to run their own nodes for more independent security and quicker verification.

9. Combining and Splitting Value

Although it would be possible to handle coins individually, it would be unwieldy to make a separate transaction for every cent in a transfer. To allow value to be split and combined, transactions contain multiple inputs and outputs. Normally there will be either a single input from a larger previous transaction or multiple inputs combining smaller amounts, and at most two outputs: one for the payment, and one returning the change, if any, back to the sender.



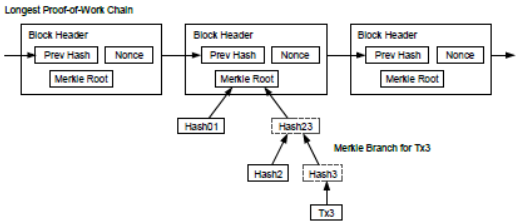
It should be noted that fan-out, where a transaction depends on several transactions, and those transactions depend on many more, is not a problem here. There is never the need to extract a complete standalone copy of a transaction's history.

10. Privacy

The traditional banking model achieves a level of privacy by limiting access to information to the parties involved and the trusted third party. The necessity to announce all transactions publicly precludes this method, but privacy can still be maintained by breaking the flow of information in another place: by keeping public keys anonymous. The public can see that someone is sending

8. Simplified Payment Verification

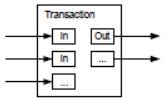
It is possible to verify payments without running a full network node. A user only needs to keep a copy of the block headers of the longest proof-of-work chain, which he can get by querying network nodes until he's convinced he has the longest chain, and obtain the Merkle branch linking the transaction to the block it's timestamped in. He can't check the transaction for himself, but by linking it to a place in the chain, he can see that a network node has accepted it, and blocks added after it further confirm the network has accepted it.



As such, the verification is reliable as long as honest nodes control the network, but is more vulnerable if the network is overpowered by an attacker. While network nodes can verify transactions for themselves, the simplified method can be fooled by an attacker's fabricated transactions for as long as the attacker can continue to overpower the network. One strategy to protect against this would be to accept alerts from network nodes when they detect an invalid block, prompting the user's software to download the full block and alerted transactions to confirm the inconsistency. Businesses that receive frequent payments will probably still want to run their own nodes for more independent security and quicker verification.

9. Combining and Splitting Value

Although it would be possible to handle coins individually, it would be unwieldy to make a separate transaction for every cent in a transfer. To allow value to be split and combined, transactions contain multiple inputs and outputs. Normally there will be either a single input from a larger previous transaction or multiple inputs combining smaller amounts, and at most two outputs: one for the payment, and one returning the change, if any, back to the sender.



It should be noted that fan-out, where a transaction depends on several transactions, and those transactions depend on many more, is not a problem here. There is never the need to extract a complete standalone copy of a transaction's history.

1320

1321

Side 6:

Ingen forskjeller

Side 7:

Ingen forskjeller

Side 8:

Ingen forskjeller

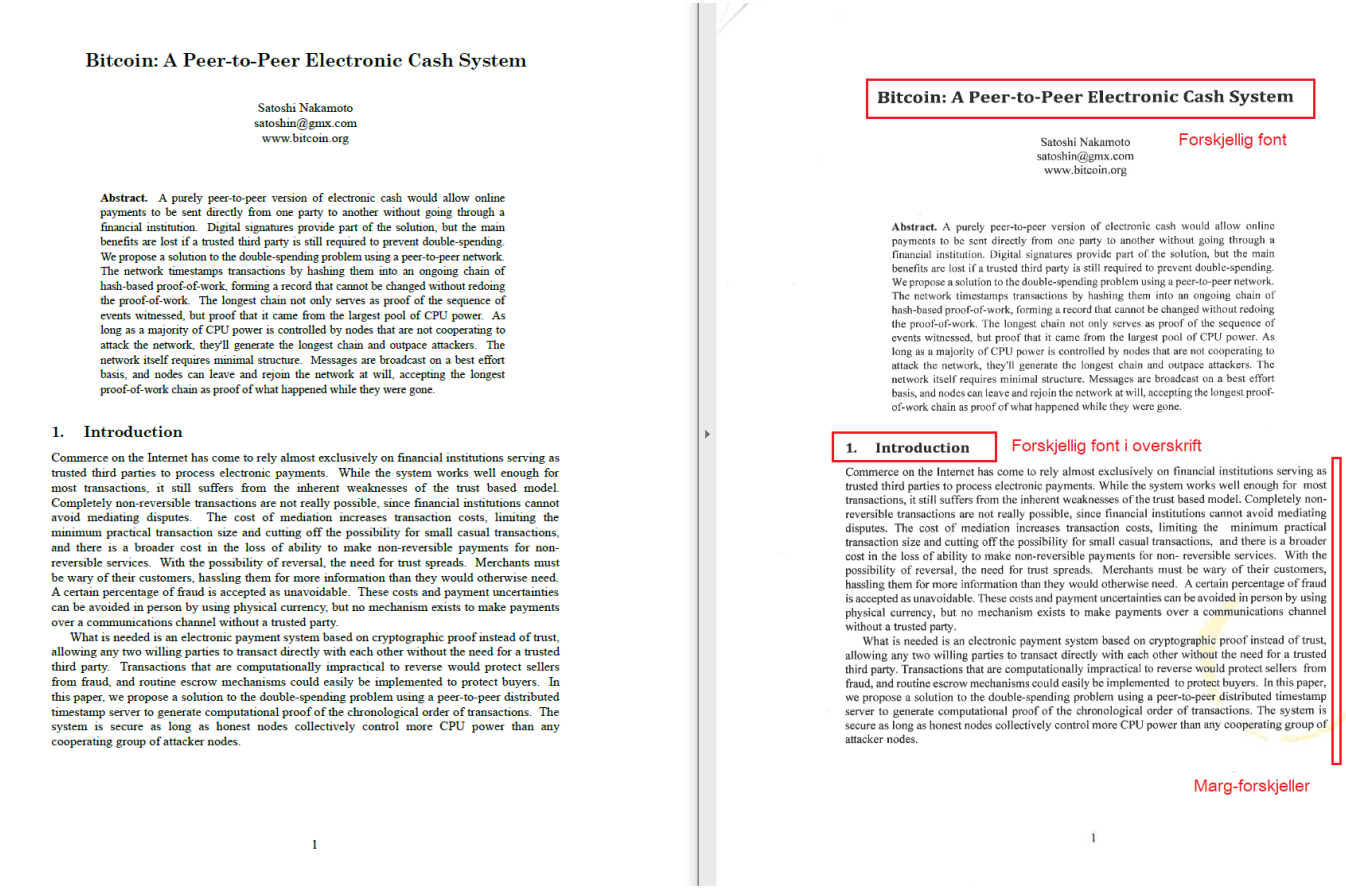
1321

1322

4.18 Vedlegg 18: Innholdsanalyse «Bilag 20.pdf»

4.18.1 Tekstforskjeller fra «Bilag 20.pdf» vs. «bitcoin.pdf»

Side 1 - «bitcoin.pdf» til venstre, «Bilag 20.pdf» til høyre:



Bitcoin: A Peer-to-Peer Electronic Cash System

Satoshi Nakamoto
satoshin@gmx.com
www.bitcoin.org

Abstract. A purely peer-to-peer version of electronic cash would allow online payments to be sent directly from one party to another without going through a financial institution. Digital signatures provide part of the solution, but the main benefits are lost if a trusted third party is still required to prevent double-spending. We propose a solution to the double-spending problem using a peer-to-peer network. The network timestamps transactions by hashing them into an ongoing chain of hash-based proof-of-work, forming a record that cannot be changed without redoing the proof-of-work. The longest chain not only serves as proof of the sequence of events witnessed, but proof that it came from the largest pool of CPU power. As long as a majority of CPU power is controlled by nodes that are not cooperating to attack the network, they'll generate the longest chain and outpace attackers. The network itself requires minimal structure. Messages are broadcast on a best effort basis, and nodes can leave and rejoin the network at will, accepting the longest proof-of-work chain as proof of what happened while they were gone.

1. Introduction

Commerce on the Internet has come to rely almost exclusively on financial institutions serving as trusted third parties to process electronic payments. While the system works well enough for most transactions, it still suffers from the inherent weaknesses of the trust based model. Completely non-reversible transactions are not really possible, since financial institutions cannot avoid mediating disputes. The cost of mediation increases transaction costs, limiting the minimum practical transaction size and cutting off the possibility for small casual transactions, and there is a broader cost in the loss of ability to make non-reversible payments for non-reversible services. With the possibility of reversal, the need for trust spreads. Merchants must be wary of their customers, hassling them for more information than they would otherwise need. A certain percentage of fraud is accepted as unavoidable. These costs and payment uncertainties can be avoided in person by using physical currency, but no mechanism exists to make payments over a communications channel without a trusted party.

What is needed is an electronic payment system based on cryptographic proof instead of trust, allowing any two willing parties to transact directly with each other without the need for a trusted third party. Transactions that are computationally impractical to reverse would protect sellers from fraud, and routine escrow mechanisms could easily be implemented to protect buyers. In this paper, we propose a solution to the double-spending problem using a peer-to-peer distributed timestamp server to generate computational proof of the chronological order of transactions. The system is secure as long as honest nodes collectively control more CPU power than any cooperating group of attacker nodes.

1

Bitcoin: A Peer-to-Peer Electronic Cash System

Forskjellig font

Satoshi Nakamoto
satoshin@gmx.com
www.bitcoin.org

Abstract. A purely peer-to-peer version of electronic cash would allow online payments to be sent directly from one party to another without going through a financial institution. Digital signatures provide part of the solution, but the main benefits are lost if a trusted third party is still required to prevent double-spending. We propose a solution to the double-spending problem using a peer-to-peer network. The network timestamps transactions by hashing them into an ongoing chain of hash-based proof-of-work, forming a record that cannot be changed without redoing the proof-of-work. The longest chain not only serves as proof of the sequence of events witnessed, but proof that it came from the largest pool of CPU power. As long as a majority of CPU power is controlled by nodes that are not cooperating to attack the network, they'll generate the longest chain and outpace attackers. The network itself requires minimal structure. Messages are broadcast on a best effort basis, and nodes can leave and rejoin the network at will, accepting the longest proof-of-work chain as proof of what happened while they were gone.

1. Introduction

Forskjellig font i overskrift

Commerce on the Internet has come to rely almost exclusively on financial institutions serving as trusted third parties to process electronic payments. While the system works well enough for most transactions, it still suffers from the inherent weaknesses of the trust based model. Completely non-reversible transactions are not really possible, since financial institutions cannot avoid mediating disputes. The cost of mediation increases transaction costs, limiting the minimum practical transaction size and cutting off the possibility for small casual transactions, and there is a broader cost in the loss of ability to make non-reversible payments for non- reversible services. With the possibility of reversal, the need for trust spreads. Merchants must be wary of their customers, hassling them for more information than they would otherwise need. A certain percentage of fraud is accepted as unavoidable. These costs and payment uncertainties can be avoided in person by using physical currency, but no mechanism exists to make payments over a communications channel without a trusted party.

What is needed is an electronic payment system based on cryptographic proof instead of trust, allowing any two willing parties to transact directly with each other without the need for a trusted third party. Transactions that are computationally impractical to reverse would protect sellers from fraud, and routine escrow mechanisms could easily be implemented to protect buyers. In this paper, we propose a solution to the double-spending problem using a peer-to-peer distributed timestamp server to generate computational proof of the chronological order of transactions. The system is secure as long as honest nodes collectively control more CPU power than any cooperating group of attacker nodes.

Marg-forskjeller

1

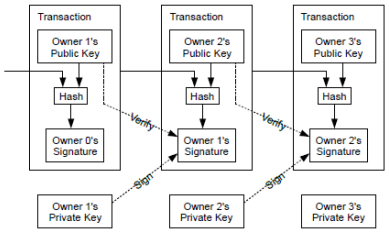
1322

1323

Side 2 - «bitcoin.pdf» til venstre, «Bilag 20.pdf» til høyre:

2. Transactions

We define an electronic coin as a chain of digital signatures. Each owner transfers the coin to the next by digitally signing a hash of the previous transaction and the public key of the next owner and adding these to the end of the coin. A payee can verify the signatures to verify the chain of ownership.

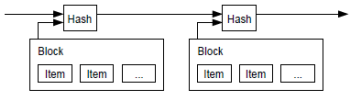


The problem of course is the payee can't verify that one of the owners did not double-spend the coin. A common solution is to introduce a trusted central authority, or mint, that checks every transaction for double spending. After each transaction, the coin must be returned to the mint to issue a new coin, and only coins issued directly from the mint are trusted not to be double-spent. The problem with this solution is that the fate of the entire money system depends on the company running the mint, with every transaction having to go through them, just like a bank.

We need a way for the payee to know that the previous owners did not sign any earlier transactions. For our purposes, the earliest transaction is the one that counts, so we don't care about later attempts to double-spend. The only way to confirm the absence of a transaction is to be aware of all transactions. In the mint based model, the mint was aware of all transactions and decided which arrived first. To accomplish this without a trusted party, transactions must be publicly announced [1], and we need a system for participants to agree on a single history of the order in which they were received. The payee needs proof that at the time of each transaction, the majority of nodes agreed it was the first received.

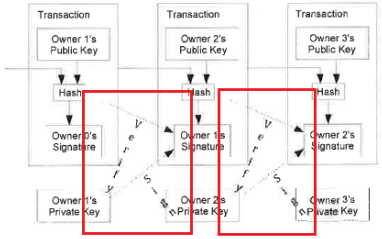
3. Timestamp Server

The solution we propose begins with a timestamp server. A timestamp server works by taking a hash of a block of items to be timestamped and widely publishing the hash, such as in a newspaper or Usenet post [2-5]. The timestamp proves that the data must have existed at the time, obviously, in order to get into the hash. Each timestamp includes the previous timestamp in its hash, forming a chain, with each additional timestamp reinforcing the ones before it.



2. Transactions

We define an electronic coin as a chain of digital signatures. Each owner transfers the coin to the next by digitally signing a hash of the previous transaction and the public key of the next owner and adding these to the end of the coin. A payee can verify the signatures to verify the chain of ownership.

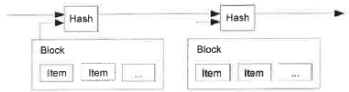


The problem of course is the payee can't verify that one of the owners did not double-spend the coin. A common solution is to introduce a trusted central authority, or mint, that checks every transaction for double spending. After each transaction, the coin must be returned to the mint to issue a new coin, and only coins issued directly from the mint are trusted not to be double-spent. The problem with this solution is that the fate of the entire money system depends on the company running the mint, with every transaction having to go through them, just like a bank.

We need a way for the payee to know that the previous owners did not sign any earlier transactions. For our purposes, the earliest transaction is the one that counts, so we don't care about later attempts to double-spend. The only way to confirm the absence of a transaction is to be aware of all transactions. In the mint based model, the mint was aware of all transactions and decided which arrived first. To accomplish this without a trusted party, transactions must be publicly announced [1], and we need a system for participants to agree on a single history of the order in which they were received. The payee needs proof that at the time of each transaction, the majority of nodes agreed it was the first received.

3. Timestamp Server

The solution we propose begins with a timestamp server. A timestamp server works by taking a hash of a block of items to be timestamped and widely publishing the hash, such as in a newspaper or Usenet post [2-5]. The timestamp proves that the data must have existed at the time, obviously, in order to get into the hash. Each timestamp includes the previous timestamp in its hash, forming a chain, with each additional timestamp reinforcing the ones before it.



1323

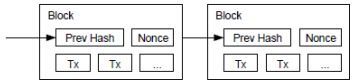
1324

Side 3 - «bitcoin.pdf» til venstre, «Bilag 20.pdf» til høyre:

4. Proof-of-Work

To implement a distributed timestamp server on a peer-to-peer basis, we will need to use a proof-of-work system similar to Adam Back's Hashcash [6], rather than newspaper or Usenet posts. The proof-of-work involves scanning for a value that when hashed, such as with SHA-256, the hash begins with a number of zero bits. The average work required is exponential in the number of zero bits required and can be verified by executing a single hash.

For our timestamp network, we implement the proof-of-work by incrementing a nonce in the block until a value is found that gives the block's hash the required zero bits. Once the CPU effort has been expended to make it satisfy the proof-of-work, the block cannot be changed without redoing the work. As later blocks are chained after it, the work to change the block would include redoing all the blocks after it.



The proof-of-work also solves the problem of determining representation in majority decision making. If the majority were based on one-IP-address-one-vote, it could be subverted by anyone able to allocate many IPs. Proof-of-work is essentially one-CPU-one-vote. The majority decision is represented by the longest chain, which has the greatest proof-of-work effort invested in it. If a majority of CPU power is controlled by honest nodes, the honest chain will grow the fastest and outpace any competing chains. To modify a past block, an attacker would have to redo the proof-of-work of the block and all blocks after it and then catch up with and surpass the work of the honest nodes. We will show later that the probability of a slower attacker catching up diminishes exponentially as subsequent blocks are added.

To compensate for increasing hardware speed and varying interest in running nodes over time, the proof-of-work difficulty is determined by a moving average targeting an average number of blocks per hour. If they're generated too fast, the difficulty increases.

5. Network

The steps to run the network are as follows:

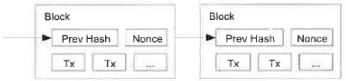
1) New transactions are broadcast to all nodes.
2) Each node collects new transactions into a block.
3) Each node works on finding a difficult proof-of-work for its block.
4) When a node finds a proof-of-work, it broadcasts the block to all nodes.
5) Nodes accept the block only if all transactions in it are valid and not already spent.
6) Nodes express their acceptance of the block by working on creating the next block in the chain, using the hash of the accepted block as the previous hash.

Nodes always consider the longest chain to be the correct one and will keep working on extending it. If two nodes broadcast different versions of the next block simultaneously, some nodes may receive one or the other first. In that case, they work on the first one they received, but save the other branch in case it becomes longer. The tie will be broken when the next proof-of-work is found and one branch becomes longer; the nodes that were working on the other branch will then switch to the longer one.

4. Proof-of-Work

To implement a distributed timestamp server on a peer-to-peer basis, we will need to use a proof-of-work system similar to Adam Back's Hashcash [6], rather than newspaper or Usenet posts. The proof-of-work involves scanning for a value that when hashed, such as with SHA-256, the hash begins with a number of zero bits. The average work required is exponential in the number of zero bits required and can be verified by executing a single hash.

For our timestamp network, we implement the proof-of-work by incrementing a nonce in the block until a value is found that gives the block's hash the required zero bits. Once the CPU effort has been expended to make it satisfy the proof-of-work, the block cannot be changed without redoing the work. As later blocks are chained after it, the work to change the block would include redoing all the blocks after it.



The proof-of-work also solves the problem of determining representation in majority decision making. If the majority were based on one-IP-address-one-vote, it could be subverted by anyone able to allocate many IPs. Proof-of-work is essentially one-CPU-one-vote. The majority decision is represented by the longest chain, which has the greatest proof-of-work effort invested in it. If a majority of CPU power is controlled by honest nodes, the honest chain will grow the fastest and outpace any competing chains. To modify a past block, an attacker would have to redo the proof-of-work of the block and all blocks after it and then catch up with and surpass the work of the honest nodes. We will show later that the probability of a slower attacker catching up diminishes exponentially as subsequent blocks are added.

To compensate for increasing hardware speed and varying interest in running nodes over time, the proof-of-work difficulty is determined by a moving average targeting an average number of blocks per hour. If they're generated too fast, the difficulty increases.

5. Network

The steps to run the network are as follows:

1) New transactions are broadcast to all nodes.
2) Each node collects new transactions into a block.
3) Each node works on finding a difficult proof-of-work for its block.
4) When a node finds a proof-of-work, it broadcasts the block to all nodes.
5) Nodes accept the block only if all transactions in it are valid and not already spent.
6) Nodes express their acceptance of the block by working on creating the next block in the chain, using the hash of the accepted block as the previous hash.

Nodes always consider the longest chain to be the correct one and will keep working on extending it. If two nodes broadcast different versions of the next block simultaneously, some nodes may receive one or the other first. In that case, they work on the first one they received, but save the other branch in case it becomes longer. The tie will be broken when the next proof- of-work is found and one branch becomes longer; the nodes that were working on the other branch will then switch to the longer one.

1324

1325

Side 4 - «bitcoin.pdf» til venstre, «Bilag 20.pdf» til høyre:

New transaction broadcasts do not necessarily need to reach all nodes. As long as they reach many nodes, they will get into a block before long. Block broadcasts are also tolerant of dropped messages. If a node does not receive a block, it will request it when it receives the next block and realizes it missed one.

6. Incentive

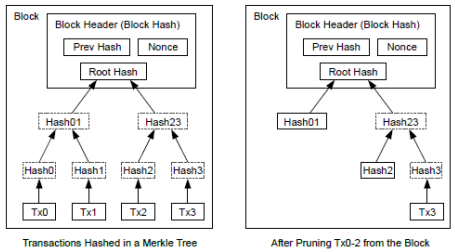
By convention, the first transaction in a block is a special transaction that starts a new coin owned by the creator of the block. This adds an incentive for nodes to support the network, and provides a way to initially distribute coins into circulation, since there is no central authority to issue them. The steady addition of a constant of amount of new coins is analogous to gold miners expending resources to add gold to circulation. In our case, it is CPU time and electricity that is expended.

The incentive can also be funded with transaction fees. If the output value of a transaction is less than its input value, the difference is a transaction fee that is added to the incentive value of the block containing the transaction. Once a predetermined number of coins have entered circulation, the incentive can transition entirely to transaction fees and be completely inflation free.

The incentive may help encourage nodes to stay honest. If a greedy attacker is able to assemble more CPU power than all the honest nodes, he would have to choose between using it to defraud people by stealing back his payments, or using it to generate new coins. He ought to find it more profitable to play by the rules, such rules that favour him with more new coins than everyone else combined, than to undermine the system and the validity of his own wealth.

7. Reclaiming Disk Space

Once the latest transaction in a coin is buried under enough blocks, the spent transactions before it can be discarded to save disk space. To facilitate this without breaking the block's hash, transactions are hashed in a Merkle Tree [7][2][5], with only the root included in the block's hash. Old blocks can then be compacted by stubbing off branches of the tree. The interior hashes do not need to be stored.



A block header with no transactions would be about 80 bytes. If we suppose blocks are generated every 10 minutes, 80 bytes * 6 * 24 * 365 = 4.2MB per year. With computer systems typically selling with 2GB of RAM as of 2008, and Moore's Law predicting current growth of 1.2GB per year, storage should not be a problem even if the block headers must be kept in memory.

New transaction broadcasts do not necessarily need to reach all nodes. As long as they reach many nodes, they will get into a block before long. Block broadcasts are also tolerant of dropped messages. If a node does not receive a block, it will request it when it receives the next block and realizes it missed one.

6. Incentive

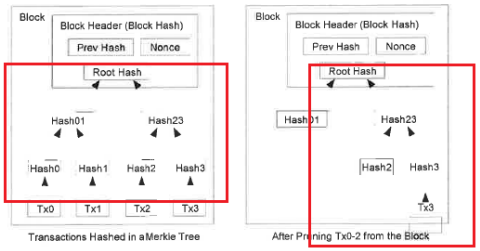
By convention, the first transaction in a block is a special transaction that starts a new coin owned by the creator of the block. This adds an incentive for nodes to support the network, and provides a way to initially distribute coins into circulation, since there is no central authority to issue them. The steady addition of a constant of amount of new coins is analogous to gold miners expending resources to add gold to circulation. In our case, it is CPU time and electricity that is expended.

The incentive can also be funded with transaction fees. If the output value of a transaction is less than its input value, the difference is a transaction fee that is added to the incentive value of the block containing the transaction. Once a predetermined number of coins have entered circulation, the incentive can transition entirely to transaction fees and be completely inflation free.

The incentive may help encourage nodes to stay honest. If a greedy attacker is able to assemble more CPU power than all the honest nodes, he would have to choose between using it to defraud people by stealing back his payments, or using it to generate new coins. He ought to find it more profitable to play by the rules, such rules that favour him with more new coins than everyone else combined, than to undermine the system and the validity of his own wealth.

7. Reclaiming Disk Space

Once the latest transaction in a coin is buried under enough blocks, the spent transactions before it can be discarded to save disk space. To facilitate this without breaking the block's hash, transactions are hashed in a Merkle Tree [7][2][5], with only the root included in the block's hash. Old blocks can then be compacted by stubbing off branches of the tree. The interior hashes do not need to be stored.



A block header with no transactions would be about 80 bytes. If we suppose blocks are generated every 10 minutes, 80 bytes * 6 * 24 * 365 = 4.2MB per year. With computer systems typically selling with 2GB of RAM as of 2008, and Moore's Law predicting current growth of 1.2GB per year, storage should not be a problem even if the block headers must be kept in memory.

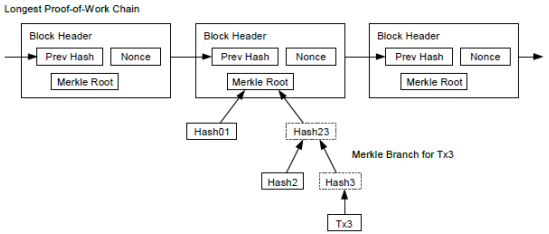
1325

1326

Side 5 - «bitcoin.pdf» til venstre, «Bilag 20.pdf» til høyre:

8. Simplified Payment Verification

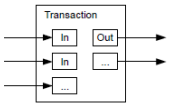
It is possible to verify payments without running a full network node. A user only needs to keep a copy of the block headers of the longest proof-of-work chain, which he can get by querying network nodes until he's convinced he has the longest chain, and obtain the Merkle branch linking the transaction to the block it's timestamped in. He can't check the transaction for himself, but by linking it to a place in the chain, he can see that a network node has accepted it, and blocks added after it further confirm the network has accepted it.



As such, the verification is reliable as long as honest nodes control the network, but is more vulnerable if the network is overpowered by an attacker. While network nodes can verify transactions for themselves, the simplified method can be fooled by an attacker's fabricated transactions for as long as the attacker can continue to overpower the network. One strategy to protect against this would be to accept alerts from network nodes when they detect an invalid block, prompting the user's software to download the full block and alerted transactions to confirm the inconsistency. Businesses that receive frequent payments will probably still want to run their own nodes for more independent security and quicker verification.

9. Combining and Splitting Value

Although it would be possible to handle coins individually, it would be unwieldy to make a separate transaction for every cent in a transfer. To allow value to be split and combined, transactions contain multiple inputs and outputs. Normally there will be either a single input from a larger previous transaction or multiple inputs combining smaller amounts, and at most two outputs: one for the payment, and one returning the change, if any, back to the sender.



It should be noted that fan-out, where a transaction depends on several transactions, and those transactions depend on many more, is not a problem here. There is never the need to extract a complete standalone copy of a transaction's history.

8. Simplified Payment Verification

It is possible to verify payments without running a full network node. A user only needs to keep a copy of the block headers of the longest proof-of-work chain, which he can get by querying network nodes until he's convinced he has the longest chain, and obtain the Merkle branch linking the transaction to the block it's timestamped in. He can't check the transaction for himself, but by linking it to a place in the chain, he can see that a network node has accepted it, and blocks added after it further confirm the network has accepted it.

Longest Proof-of-Work Chain
Block Header
Prev Hash
Nonce
Merkle Root
Block Header
Prev Hash
Nonce
Merkle Root
Block Header
Prev Hash
Nonce
Merkle Root
Hash01
Hash23
Merkle Branch for Tx3
Hash2
Hash3
Tx3

As such, the verification is reliable as long as honest nodes control the network, but is more vulnerable if the network is overpowered by an attacker. While network nodes can verify transactions for themselves, the simplified method can be fooled by an attacker's fabricated transactions for as long as the attacker can continue to overpower the network. One strategy to protect against this would be to accept alerts from network nodes when they detect an invalid block, prompting the user's software to download the full block and alerted transactions to confirm the inconsistency. Businesses that receive frequent payments will probably still want to run their own nodes for more independent security and quicker verification.

9. Combining and Splitting Value

Although it would be possible to handle coins individually, it would be unwieldy to make a separate transaction for every cent in a transfer. To allow value to be split and combined, transactions contain multiple inputs and outputs. Normally there will be either a single input from a larger previous transaction or multiple inputs combining smaller amounts, and at most two outputs: one for the payment, and one returning the change, if any, back to the sender.

Transaction
In
Out
In
...
...

It should be noted that fan-out, where a transaction depends on several transactions, and those transactions depend on many more, is not a problem here. There is never the need to extract a complete standalone copy of a transaction's history.

5

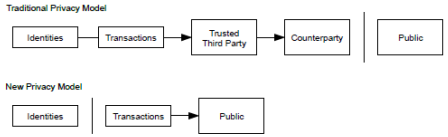
1326

1327

Side 6 - «bitcoin.pdf» til venstre, «Bilag 20.pdf» til høyre:

10. Privacy

The traditional banking model achieves a level of privacy by limiting access to information to the parties involved and the trusted third party. The necessity to announce all transactions publicly precludes this method, but privacy can still be maintained by breaking the flow of information in another place: by keeping public keys anonymous. The public can see that someone is sending an amount to someone else, but without information linking the transaction to anyone. This is similar to the level of information released by stock exchanges, where the time and size of individual trades, the "tape", is made public, but without telling who the parties were.



As an additional firewall, a new key pair should be used for each transaction to keep them from being linked to a common owner. Some linking is still unavoidable with multi-input transactions, which necessarily reveal that their inputs were owned by the same owner. The risk is that if the owner of a key is revealed, linking could reveal other transactions that belonged to the same owner.

11. Calculations

We consider the scenario of an attacker trying to generate an alternate chain faster than the honest chain. Even if this is accomplished, it does not throw the system open to arbitrary changes, such as creating value out of thin air or taking money that never belonged to the attacker. Nodes are not going to accept an invalid transaction as payment, and honest nodes will never accept a block containing them. An attacker can only try to change one of his own transactions to take back money he recently spent.

The race between the honest chain and an attacker chain can be characterized as a Binomial Random Walk. The success event is the honest chain being extended by one block, increasing its lead by +1, and the failure event is the attacker's chain being extended by one block, reducing the gap by -1.

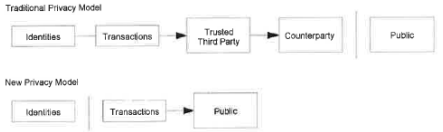
The probability of an attacker catching up from a given deficit is analogous to a Gambler's Ruin problem. Suppose a gambler with unlimited credit starts at a deficit and plays potentially an infinite number of trials to try to reach breakeven. We can calculate the probability he ever reaches breakeven, or that an attacker ever catches up with the honest chain, as follows [8]:

p = probability an honest node finds the next block
q = probability the attacker finds the next block
q_z = probability the attacker will ever catch up from z blocks behind

$$q_z = \begin{cases} 1 & \text{if } p \le q \\ (q/p)^z & \text{if } p > q \end{cases}$$

10. Privacy

The traditional banking model achieves a level of privacy by limiting access to information to the parties involved and the trusted third party. The necessity to announce all transactions publicly precludes this method, but privacy can still be maintained by breaking the flow of information in another place: by keeping public keys anonymous. The public can see that someone is sending an amount to someone else, but without information linking the transaction to anyone. This is similar to the level of information released by stock exchanges, where the time and size of individual trades, the "tape", is made public, but without telling who the parties were.



As an additional firewall, a new key pair should be used for each transaction to keep them from being linked to a common owner. Some linking is still unavoidable with multi-input transactions, which necessarily reveal that their inputs were owned by the same owner. The risk is that if the owner of a key is revealed, linking could reveal other transactions that belonged to the same owner.

11. Calculations

We consider the scenario of an attacker trying to generate an alternate chain faster than the honest chain. Even if this is accomplished, it does not throw the system open to arbitrary changes, such as creating value out of thin air or taking money that never belonged to the attacker. Nodes are not going to accept an invalid transaction as payment, and honest nodes will never accept a block containing them. An attacker can only try to change one of his own transactions to take back money he recently spent.

The race between the honest chain and an attacker chain can be characterized as a Binomial Random Walk. The success event is the honest chain being extended by one block, increasing its lead by +1, and the failure event is the attacker's chain being extended by one block, reducing the gap by -1.

The probability of an attacker catching up from a given deficit is analogous to a Gambler's Ruin problem. Suppose a gambler with unlimited credit starts at a deficit and plays potentially an infinite number of trials to try to reach breakeven. We can calculate the probability he ever reaches breakeven, or that an attacker ever catches up with the honest chain, as follows [8]:

p = probability an honest node finds the next block
q = probability the attacker finds the next block
q_z = probability the attacker will ever catch up from z blocks behind

$$q_z = \begin{cases} 1 & \text{if } p \le q \\ (q/p)^z & \text{if } p < q \end{cases}$$

Forskjellig symbol

1327

1328

Side 7 - «bitcoin.pdf» til venstre, «Bilag 20.pdf» til høyre:

Given our assumption that $p > q$, the probability drops exponentially as the number of blocks the attacker has to catch up with increases. With the odds against him, if he doesn't make a lucky lunge forward early on, his chances become vanishingly small as he falls further behind.

We now consider how long the recipient of a new transaction needs to wait before being sufficiently certain the sender can't change the transaction. We assume the sender is an attacker who wants to make the recipient believe he paid him for a while, then switch it to pay back to himself after some time has passed. The receiver will be alerted when that happens, but the sender hopes it will be too late.

The receiver generates a new key pair and gives the public key to the sender shortly before signing. This prevents the sender from preparing a chain of blocks ahead of time by working on it continuously until he is lucky enough to get far enough ahead, then executing the transaction at that moment. Once the transaction is sent, the dishonest sender starts working in secret on a parallel chain containing an alternate version of his transaction.

The recipient waits until the transaction has been added to a block and z blocks have been linked after it. He doesn't know the exact amount of progress the attacker has made, but assuming the honest blocks took the average expected time per block, the attacker's potential progress will be a Poisson distribution with expected value:

$$\lambda = z\frac{q}{p}$$

To get the probability the attacker could still catch up now, we multiply the Poisson density for each amount of progress he could have made by the probability he could catch up from that point:

$$\sum_{k=0}^{\infty} \frac{\lambda^k e^{-\lambda}}{k!} \cdot \begin{cases} (q/p)^{(z-k)} & \text{if } k \le z \\ 1 & \text{if } k > z \end{cases}$$

Rearranging to avoid summing the infinite tail of the distribution...

$$1 - \sum_{k=0}^{z} \frac{\lambda^k e^{-\lambda}}{k!} \left(1 - (q/p)^{(z-k)}\right)$$

Converting to C code...

```
#include <math.h>
double AttackerSuccessProbability(double q, int z)
{
    double p = 1.0 - q;
    double lambda = z * (q / p);
    double sum = 1.0;
    int i, k;
    for (k = 0; k <= z; k++)
    {
        double poisson = exp(-lambda);
        for (i = 1; i <= k; i++)
            poisson *= lambda / i;
        sum -= poisson * (1 - pow(q / p, z - k));
    }
    return sum;
}
```


Given our assumption that $p > q$, the probability drops exponentially as the number of blocks the attacker has to catch up with increases. With the odds against him, if he doesn't make a lucky lunge forward early on, his chances become vanishingly small as he falls further behind.

We now consider how long the recipient of a new transaction needs to wait before being sufficiently certain the sender can't change the transaction. We assume the sender is an attacker who wants to make the recipient believe he paid him for a while, then switch it to pay back to himself after some time has passed. The receiver will be alerted when that happens, but the sender hopes it will be too late.

The receiver generates a new key pair and gives the public key to the sender shortly before signing. This prevents the sender from preparing a chain of blocks ahead of time by working on it continuously until he is lucky enough to get far enough ahead, then executing the transaction at that moment. Once the transaction is sent, the dishonest sender starts working in secret on a parallel chain containing an alternate version of his transaction.

The recipient waits until the transaction has been added to a block and z blocks have been linked after it. He doesn't know the exact amount of progress the attacker has made, but assuming the honest blocks took the average expected time per block, the attacker's potential progress will be a Poisson distribution with expected value:

$$\Theta = z\frac{q}{p}$$

To get the probability the attacker could still catch up now, we multiply the Poisson density for each amount of progress he could have made by the probability he could catch up from that point:

$$\sum_{k=0}^{\infty} \frac{\Theta^k e^{-\Theta}}{k!} \begin{cases} (q/p)^{(z-k)} & \text{if } k \le z \\ 1 & \text{if } k < z \end{cases}$$

Forskjellig symbol

Rearranging to avoid summing the infinite tail of the distribution...

$$1 - \sum_{k=0}^{z} \frac{\Theta^k e^{-\Theta}}{k!} \left(1 - (q/p)^{(z-k)}\right)$$

Converting to C code...

```
#include <math.h>
double AttackerSuccessProbability(double q, int z)
{
    double p = 1.0 - q;
    double lambda = z * (q / p);
    double sum = 1.0;
    int i, k;
    for (k = 0; k <= z; k++)
    {
        double poisson = exp(-lambda);
        for (i = 1; i <= k; i++)
            poisson *= lambda / i;
        sum -= poisson * (1 - pow(q / p, z - k));
    }
    return sum;
}
```


1328

1329

Side 8 - «bitcoin.pdf» til venstre, «Bilag 20.pdf» til høyre:

Running some results, we can see the probability drop off exponentially with z.

```
q=0.1
z=0     P=1.0000000
z=1     P=0.2045873
z=2     P=0.0509779
z=3     P=0.0131722
z=4     P=0.0034552
z=5     P=0.0009137
z=6     P=0.0002428
z=7     P=0.0000647
z=8     P=0.0000173
z=9     P=0.0000046
z=10    P=0.0000012

q=0.3
z=0     P=1.0000000
z=5     P=0.1773523
z=10    P=0.0416605
z=15    P=0.0101008
z=20    P=0.0024804
z=25    P=0.0006132
z=30    P=0.0001522
z=35    P=0.0000379
z=40    P=0.0000095
z=45    P=0.0000024
z=50    P=0.0000006
```

Solving for P less than 0.1%...

```
P < 0.001
q=0.10   z=5
q=0.15   z=8
q=0.20   z=11
q=0.25   z=15
q=0.30   z=24
q=0.35   z=41
q=0.40   z=89
q=0.45   z=340
```

12. Conclusion

We have proposed a system for electronic transactions without relying on trust. We started with the usual framework of coins made from digital signatures, which provides strong control of ownership, but is incomplete without a way to prevent double-spending. To solve this, we proposed a peer-to-peer network using proof-of-work to record a public history of transactions that quickly becomes computationally impractical for an attacker to change if honest nodes control a majority of CPU power. The network is robust in its unstructured simplicity. Nodes work all at once with little coordination. They do not need to be identified, since messages are not routed to any particular place and only need to be delivered on a best effort basis. Nodes can leave and rejoin the network at will, accepting the proof-of-work chain as proof of what happened while they were gone. They vote with their CPU power, expressing their acceptance of valid blocks by working on extending them and rejecting invalid blocks by refusing to work on them. Any needed rules and incentives can be enforced with this consensus mechanism.

8

Running some results, we can see the probability drop off exponentially with z.

```
q=0.1
z=0     P=1.0000000
z=1     P=0.2045873
z=2     P=0.0509779
z=3     P=0.0131722
z=4     P=0.0034552
z=5     P=0.0009137
z=6     P=0.0002428
z=7     P=0.0000647
z=8     P=0.0000173
z=9     P=0.0000046
z=10    P=0.0000012

q=0.3
z=0     P=1.0000000
z=5     P=0.1773523
z=10    P=0.0416605
z=15    P=0.0101008
z=20    P=0.0024804
z=25    P=0.0006132
z=30    P=0.0001522
z=35    P=0.0000379
z=40    P=0.0000095
z=45    P=0.0000024
z=50    P=0.0000006
```

Solving for P less than 0.1%...

```
P < 0.001
q=0.10   z=5
q=0.15   z=8
q=0.20   z=11
q=0.25   z=15
q=0.30   z=24
q=0.35   z=41
q=0.40   z=89
q=0.45   z=340
```

12. Conclusion

We have proposed a system for electronic transactions without relying on trust. We started with the usual framework of coins made from digital signatures, which provides strong control of ownership, but is incomplete without a way to prevent double-spending. To solve this, we proposed a peer-to-peer network using proof-of-work to record a public history of transactions that quickly becomes computationally impractical for an attacker to change if honest nodes control a majority of CPU power. The network is robust in its unstructured simplicity. Nodes work all at once with little coordination. They do not need to be identified, since messages are not routed to any particular place and only need to be delivered on a best effort basis. Nodes can leave and rejoin the network at will, accepting the proof-of-work chain as proof of what happened while they were gone. They vote with their CPU power, expressing their acceptance of valid blocks by working on extending them and rejecting invalid blocks by refusing to work on them. Any needed rules and incentives can be enforced with this consensus mechanism.

8

1329

1330

Side 9 - «bitcoin.pdf» til venstre, «Bilag 20.pdf» til høyre:

References

[1] W. Dai, "b-money," http://www.weidai.com/bmoney.txt, 1998.

[2] H. Massias, X.S. Avila, and J.-J. Quisquater, "Design of a secure timestamping service with minimal trust requirements," In *20th Symposium on Information Theory in the Benelux*, May 1999.

[3] S. Haber, W.S. Stornetta, "How to time-stamp a digital document," In *Journal of Cryptology*, vol 3, no 2, pages 99-111, 1991.

[4] D. Bayer, S. Haber, W.S. Stornetta, "Improving the efficiency and reliability of digital time-stamping," In *Sequences II: Methods in Communication, Security and Computer Science*, pages 329-334, 1993.

[5] S. Haber, W.S. Stornetta, "Secure names for bit-strings," In *Proceedings of the 4th ACM Conference on Computer and Communications Security*, pages 28-35, April 1997.

[6] A. Back, "Hashcash - a denial of service counter-measure," http://www.hashcash.org/papers/hashcash.pdf, 2002.

[7] R.C. Merkle, "Protocols for public key cryptosystems," In *Proc. 1980 Symposium on Security and Privacy*, IEEE Computer Society, pages 122-133, April 1980.

[8] W. Feller, "An introduction to probability theory and its applications," 1957.

References

[1] W. Dai, "b-money," http://www.weidai.com/bmoney.txt, 1998.

[2] H. Massias, X.S. Avila, and J.-J. Quisquater, "Design of a secure timestamping service with minimal trust requirements," In *20th Symposium on Information Theory in the Benelux*, May 1999.

[3] S. Haber, W.S. Stornetta, "How to time-stamp a digital document," In *Journal of Cryptology*, vol 3, no 2, pages 99-111, 1991.

[4] D. Bayer, S. Haber, W.S. Stornetta, "Improving the efficiency and reliability of digital time-stamping," In *Sequences II: Methods in Communication, Security and Computer Science*, pages 329-334, 1993.

[5] S. Haber, W.S. Stornetta, "Secure names for bit-strings," In *Proceedings of the 4th ACM Conference on Computer and Communications Security*, pages 28-35, April 1997.

[6] A. Back, "Hashcash - a denial of service counter-measure," http://www.hashcash.org/papers/hashcash.pdf, 2002.

[7] R.C. Merkle, "Protocols for public key cryptosystems," In *Proc. 1980 Symposium on Security and Privacy*, IEEE Computer Society, pages 122-133, April 1980.

[8] W. Feller, "An introduction to probability theory and its applications," 1957.

1330

4.19 Vedlegg 19: Innholdsanalyse «Bilag 26.pdf»

4.19.1 Forskjeller forside

«J-stor-original-article.pdf»:

Tominaga Nakamoto, 1715-46. A Tokugawa Iconoclast
Author(s): Katō Shūichi
Source: *Monumenta Nipponica*, 1967, Vol. 22, No. 1/2 (1967), pp. 177-193
Published by: Sophia University
Stable URL: https://www.jstor.org/stable/2383230

«Bilag26.pdf»:

Tominaga Nakamoto, 1715-46. A Tokugawa Iconoclast
Author(s): Katō Shūichi
Source: *Monumenta Nipponica*, Vol. 22, No. 1/2 (1967), pp. 177-193
Published by: Sophia University
Stable URL: http://www.jstor.org/stable/2383230
Accessed: 05/01/2008 11:17

«1967-kato.pdf»:

4.19.2 Forskjeller bunntekst forside

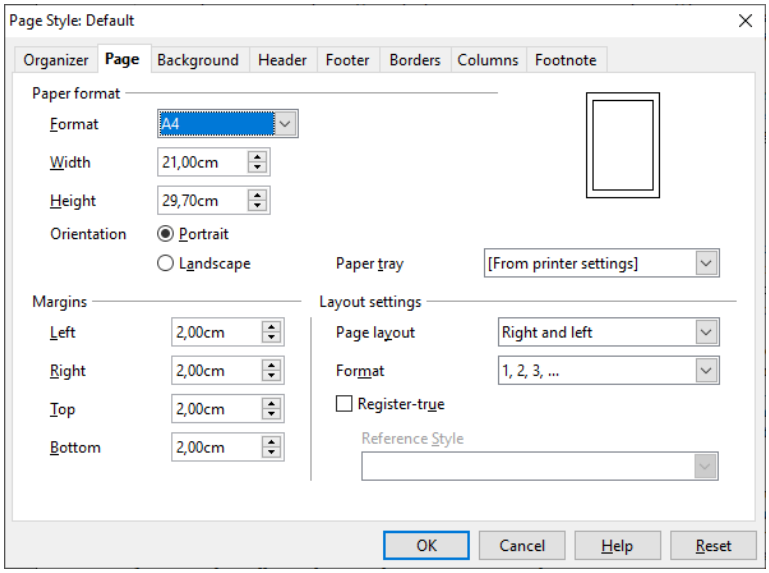
«J-stor-original-article.pdf»

«Bilag26.pdf»

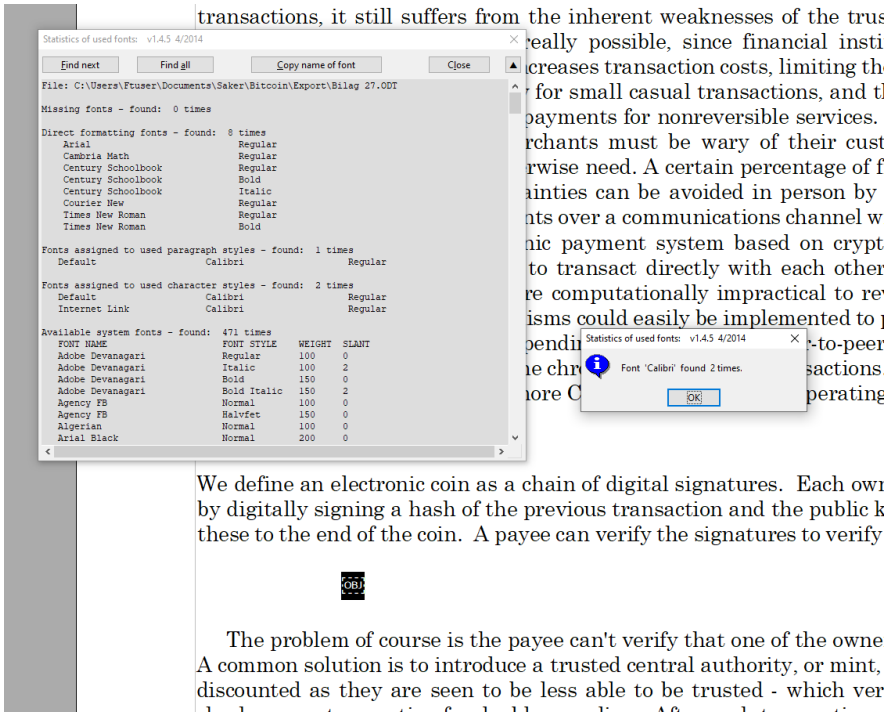
«1967-kato.pdf»

4.20 Vedlegg 20: Innholdsanalyse «Bilag 27.odt»

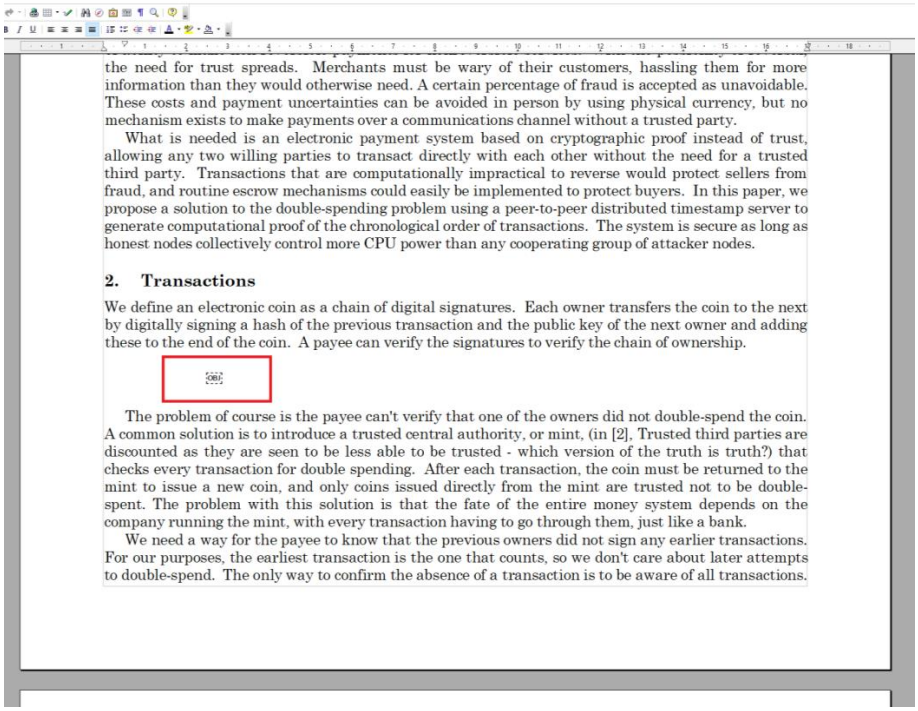
4.20.1 Papirstørrelse og marger



4.20.2 Fonter fra Open Office extension:



4.20.3 Manglende objekt:



the need for trust spreads. Merchants must be wary of their customers, hassling them for more information than they would otherwise need. A certain percentage of fraud is accepted as unavoidable. These costs and payment uncertainties can be avoided in person by using physical currency, but no mechanism exists to make payments over a communications channel without a trusted party.

What is needed is an electronic payment system based on cryptographic proof instead of trust, allowing any two willing parties to transact directly with each other without the need for a trusted third party. Transactions that are computationally impractical to reverse would protect sellers from fraud, and routine escrow mechanisms could easily be implemented to protect buyers. In this paper, we propose a solution to the double-spending problem using a peer-to-peer distributed timestamp server to generate computational proof of the chronological order of transactions. The system is secure as long as honest nodes collectively control more CPU power than any cooperating group of attacker nodes.

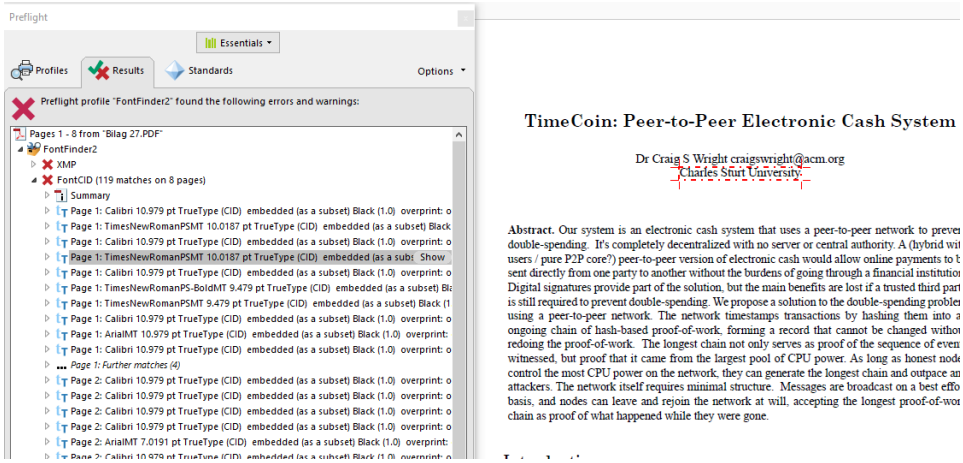
2. Transactions

We define an electronic coin as a chain of digital signatures. Each owner transfers the coin to the next by digitally signing a hash of the previous transaction and the public key of the next owner and adding these to the end of the coin. A payee can verify the signatures to verify the chain of ownership.

The problem of course is the payee can't verify that one of the owners did not double-spend the coin. A common solution is to introduce a trusted central authority, or mint, (in [2], Trusted third parties are discounted as they are seen to be less able to be trusted - which version of the truth is truth?) that checks every transaction for double spending. After each transaction, the coin must be returned to the mint to issue a new coin, and only coins issued directly from the mint are trusted not to be double-spent. The problem with this solution is that the fate of the entire money system depends on the company running the mint, with every transaction having to go through them, just like a bank.

We need a way for the payee to know that the previous owners did not sign any earlier transactions. For our purposes, the earliest transaction is the one that counts, so we don't care about later attempts to double-spend. The only way to confirm the absence of a transaction is to be aware of all transactions.

4.20.4 Font-type



4.20.5 Tekstforskjeller fra «bitcoin.pdf» vs. «Bilag 27.odt»

Her vises kun «Bilag 27.odt» i PDF-format. All «endret» tekst er uthevet i gult, all tillegg av tekst er uthevet i blått og andre endringer er markert med et hvitt kryss på rød bakgrunn.

Side 1

TimeCoin: Peer-to-Peer Electronic Cash System

Dr Craig S Wright craigswright@acm.org
Charles Sturt University

Abstract. Our system is an electronic cash system that uses a peer-to-peer network to prevent double-spending. It's completely decentralized with no server or central authority. A (hybrid with users / pure P2P core?) peer-to-peer version of electronic cash would allow online payments to be sent directly from one party to another without the burdens of going through a financial institution. Digital signatures provide part of the solution, but the main benefits are lost if a trusted third party is still required to prevent double-spending. We propose a solution to the double-spending problem using a peer-to-peer network. The network timestamps transactions by hashing them into an ongoing chain of hash-based proof-of-work, forming a record that cannot be changed without redoing the proof-of-work. The longest chain not only serves as proof of the sequence of events witnessed, but proof that it came from the largest pool of CPU power. As long as honest nodes control the most CPU power on the network, they can generate the longest chain and outpace any attackers. The network itself requires minimal structure. Messages are broadcast on a best effort basis, and nodes can leave and rejoin the network at will, accepting the longest proof-of-work chain as proof of what happened while they were gone.

1. Introduction

Commerce on the Internet has come to rely almost exclusively on financial institutions serving as trusted third parties to process electronic payments. While the system works well enough for most transactions, it still suffers from the inherent weaknesses of the trust based model. Completely non-reversible transactions are not really possible, since financial institutions cannot avoid mediating disputes. The cost of mediation increases transaction costs, limiting the minimum practical transaction size and cutting off the possibility for small casual transactions, and there is a broader cost in the loss of ability to make non-reversible payments for nonreversible services. With the possibility of reversal, the need for trust spreads. Merchants must be wary of their customers, hassling them for more information than they would otherwise need. A certain percentage of fraud is accepted as unavoidable. These costs and payment uncertainties can be avoided in person by using physical currency, but no mechanism exists to make payments over a communications channel without a trusted party.

What is needed is an electronic payment system based on cryptographic proof instead of trust, allowing any two willing parties to transact directly with each other without the need for a trusted third party. Transactions that are computationally impractical to reverse would protect sellers from fraud, and routine escrow mechanisms could easily be implemented to protect buyers. In this paper, we propose a solution to the double-spending problem using a peer-to-peer distributed timestamp server to generate computational proof of the chronological order of transactions. The system is secure as long as honest nodes collectively control more CPU power than any cooperating group of attacker nodes.

2. Transactions

We define an electronic coin as a chain of digital signatures. Each owner transfers the coin to the next by digitally signing a hash of the previous transaction and the public key of the next owner and adding these to the end of the coin. A payee can verify the signatures to verify the chain of ownership.

The problem of course is the payee can't verify that one of the owners did not double-spend the coin. A common solution is to introduce a trusted central authority, or mint, (in [2], Trusted third parties are discounted as they are seen to be less able to be trusted - which version of the truth is

Side 2

truth?) that checks every transaction for double spending. After each transaction, the coin must be returned to the mint to issue a new coin, and only coins issued directly from the mint are trusted not to be double-spent. The problem with this solution is that the fate of the entire money system depends on the company running the mint, with every transaction having to go through them, just like a bank.

We need a way for the payee to know that the previous owners did not sign any earlier transactions. For our purposes, the earliest transaction is the one that counts, so we don't care about later attempts to double-spend. The only way to confirm the absence of a transaction is to be aware of all transactions. In the mint based model, the mint was aware of all transactions and decided which arrived first. To accomplish this without a trusted party, transactions must be publicly announced [1], and we need a system for participants to agree on a single history of the order in which they were received. The payee needs proof that at the time of each transaction, the majority of nodes agreed it was the first received.

3. Timestamp Server

The solution we propose begins with a timestamp server. A timestamp server works by taking a hash of a block of items to be timestamped and widely publishing the hash, such as in a "newspaper or Usenet post" [2]. The timestamp is admissible evidence demonstrating that the data must have existed at the time, obviously, in order to get into the hash. Each timestamp includes the previous timestamp in its hash, forming a chain, with each additional timestamp reinforcing the ones before it.

See [2] - add a simple diagram based on ref 2 - no binary tree.

Need to explain, SPV and binary trees are to scale

[2] defines the timechain - we extend this noting that his rounds are published as blocks - we use a distributed publiscation to ensure many see the file digest

4. Proof-of-Work

To implement a distributed timestamp server on a peer-to-peer basis, we will need to use a proofof-work system similar to Adam Back's Hashcash [6], rather than newspaper or Usenet posts. The proof-of-work involves scanning for a value that when hashed, such as with SHA-256, the hash begins with a number of zero bits. The average work required is exponential in the number of zero bits required and can be verified by executing a single hash.

For our timestamp network, we implement the proof-of-work by incrementing a nonce in the block until a value is found that gives the block's hash the required zero bits. Once the CPU effort has been expended to make it satisfy the proof-of-work, the block cannot be changed without redoing the work. As later blocks are chained after it, the work to change the block would include redoing all the blocks after it.

See [2] - add a simple diagram based on ref 2

See Mojo Nation and PPay

The proof-of-work also solves the problem of determining representation in majority decision making. If the majority were based on one-IP-address-one-vote, it could be subverted by anyone able to allocate many IPs. Proof-of-work is essentially one-CPU-one-vote. The majority decision is represented by the longest chain, which has the greatest proof-of-work effort invested in it. If a majority of CPU power is controlled by honest nodes, the honest chain will grow the fastest and outpace any competing chains. To modify a past block, an attacker would have to redo the proof-of-work of the block and all blocks after it and then catch up with and surpass the work of the honest nodes. We will show later that the probability of a slower attacker catching up diminishes exponentially as subsequent blocks are added.

To compensate for increasing hardware speed and varying interest in running nodes over time, the proof-of-work difficulty is determined by a moving average targeting an average number of blocks per hour. If they're generated too fast, the difficulty increases.

5. Network

The steps to run the network are as follows:

1. New transactions are broadcast to all nodes.
2. Each node collects new transactions into a block.
3. Each node works on finding a difficult proof-of-work for its block.
4. When a node finds a proof-of-work, it broadcasts the block to all nodes.
5. Nodes accept the block only if all transactions in it are valid and not already spent.
6. Nodes express their acceptance of the block by working on creating the next block in the chain, using the hash of the accepted block as the previous hash.

Nodes always consider the longest chain to be the correct one and will keep working on extending it. If two nodes broadcast different versions of the next block simultaneously, some nodes may receive one or the other first. In that case, they work on the first one they received, but save the other branch in case it becomes longer. The tie will be broken when the next proofof-work is found and one branch becomes longer; the nodes that were working on the other branch will then switch to the longer one.

New transaction broadcasts do not necessarily need to reach all nodes. As long as they reach many nodes, they will get into a block before long. Block broadcasts are also tolerant of dropped messages. If a node does not receive a block, it will request it when it receives the next block and realises it missed one.

6. Incentive

By convention, the first transaction in a block is a special transaction that starts a new coin owned by the creator of the block. This adds an incentive for nodes to support the network, and provides a way to initially distribute coins into circulation, since there is no central authority to issue them. The steady addition of a constant of amount of new coins is analogous to gold miners expending resources to add gold to circulation. In our case, it is CPU time and electricity that is expended.

The incentive can also be funded with transaction fees. If the output value of a transaction is less than its input value, the difference is a transaction fee that is added to the incentive value of the block containing the transaction. Once a predetermined number of coins have entered circulation, the subsidy will be removed and the incentive shall be transitioned entirely to transaction fees and be in the form an inflationless commodity system.

The incentive may help encourage nodes to stay honest. If a greedy attacker is able to assemble more CPU power than all the honest nodes, he would have to choose between using it to defraud people by stealing back his payments, or using it to generate new coins. He ought to find it more profitable to play by the rules, such rules that favour him with more new coins than everyone else combined, than to undermine the system and the validity of his own wealth.

NOTE: LLM - "attacker" is defined. The system has its own evidence trail to allow honest systems to ensure dishonest ones cannot win.

Side 4

To compensate for increasing hardware speed and varying interest in running nodes over time, the proof-of-work difficulty is determined by a moving average targeting an average number of blocks per hour. If they're generated too fast, the difficulty increases.

7. Reclaiming Disk Space

Once the latest transaction in a coin is buried under enough blocks, the spent transactions before it can be discarded to save disk space. To facilitate this without breaking the block's hash, transactions are hashed in a Binary Tree [7][2][5], with only the root included in the block's hash. Old blocks can then be compacted by stubbing off branches of the tree. The interior hashes do not need to be stored.

A block header with no transactions would be about 80 bytes. If we suppose blocks are generated every 10 minutes, 80 bytes * 6 * 24 * 365 = 4.2MB per year. With computer systems typically selling with 2GB of RAM as of 2008, and Moore's Law predicting current growth of 1.2GB per year, storage should not be a problem even if the block headers must be kept in memory.

8. Simplified Payment Verification

It is possible to verify payments without running a full network node. A user only needs to keep a copy of the block headers of the longest proof-of-work chain, which he can get by querying network nodes until he's convinced he has the longest chain, and obtain the Merkle branch linking the transaction to the block it's timestamped in. He can't check the transaction for himself, but by linking it to a place in the chain, he can see that a network node has accepted it, and blocks added after it further confirm the network has accepted it.

Doc [2] hasha simple full timechain model - need to have this extended as image with blocks to reference "rounds"

As such, the verification is reliable as long as honest nodes control the network, but is more vulnerable if the network is overpowered by an attacker. While network nodes can verify transactions for themselves and are only vulnerable to reversal, the simplified method can be

fooled by an attacker's fabricated transactions for as long as the attacker can continue to overpower the network. One strategy to protect against this would be to accept alerts from network nodes when they detect an invalid block, prompting the user's software to download the full block and alerted transactions to confirm the inconsistency. Businesses that receive frequent payments will probably still want to run their own nodes for more independent security and quicker verification.

9. Combining and Splitting Value

Although it would be possible to handle coins individually, it would be unwieldy to make a separate transaction for every cent in a transfer. To allow value to be split and combined, transactions contain multiple inputs and outputs. Normally there will be either a single input from a larger previous transaction or multiple inputs combining smaller amounts, and at most two outputs: one for the payment, and one returning the change, if any, back to the sender.

It should be noted that fan-out, where a transaction depends on several transactions, and those transactions depend on many more, is not a problem here. There is never the need to extract a complete standalone copy of a transaction's history.

10. Privacy

The traditional banking model achieves a level of privacy by limiting access to information to the parties involved and the trusted third party. The necessity to announce all transactions publicly precludes this method, but privacy can still be maintained by breaking the flow of information in another place: by keeping public keys anonymous. The public can see that someone is sending an amount to someone else, but without information linking the transaction to anyone. This is similar to the level of information released by stock exchanges, where the time and size of individual trades, the "tape", is made public, but without telling who the parties were.

As an additional firewall, a new key pair should be used for each transaction to keep them from being linked to a common owner. Some linking is still unavoidable with multi-input transactions, which necessarily reveal that their inputs were owned by the same owner. The risk is that if the owner of a key is revealed, linking could reveal other transactions that belonged to the same owner.

11. Calculations

We consider the scenario of an attacker trying to generate an alternate chain faster than the honest chain. Even if this is accomplished, it does not throw the system open to arbitrary changes, such as creating value out of thin air or taking money that never belonged to the attacker. Nodes are not going to accept an invalid transaction as payment, and honest nodes will never accept a block containing them. An attacker can only try to change one of his own transactions to take back money he recently spent.

The race between the honest chain and an attacker chain can be characterised as a Binomial Random Walk. The success event is the honest chain being extended by one block, increasing its lead by +1, and the failure event is the attacker's chain being extended by one block, reducing the gap by -1.

The probability of an attacker catching up from a given deficit is analogous to a Gambler's Ruin problem. Suppose a gambler with unlimited credit starts at a deficit and plays potentially an infinite number of trials to try to reach breakeven. We can calculate the probability he ever reaches breakeven, or that an attacker ever catches up with the honest chain, as follows [8]:

Side 6

p = probability an honest node finds the next block q = probability the attacker finds the next block
q_z = probability the attacker will ever catch up from z blocks behind

Given our assumption that $p > q$, the probability drops exponentially as the number of blocks the attacker has to catch up with increases. With the odds against him, if he doesn't make a lucky lunge forward early on, his chances become vanishingly small as he falls further behind.

We now consider how long the recipient of a new transaction needs to wait before being sufficiently certain the sender can't change the transaction. We assume the sender is an attacker who wants to make the recipient believe he paid him for a while, then switch it to pay back to himself after some time has passed. The receiver will be alerted when that happens, but the sender hopes it will be too late.

The receiver generates a new key pair and gives the public key to the sender shortly before signing. This prevents the sender from preparing a chain of blocks ahead of time by working on it continuously until he is lucky enough to get far enough ahead, then executing the transaction at that moment. Once the transaction is sent, the dishonest sender starts working in secret on a parallel chain containing an alternate version of his transaction.

The recipient waits until the transaction has been added to a block and z blocks have been linked after it. He doesn't know the exact amount of progress the attacker has made, but assuming the honest blocks took the average expected time per block, the attacker's potential progress will be a Poisson distribution with expected value:

To get the probability the attacker could still catch up now, we multiply the Poisson density for each amount of progress he could have made by the probability he could catch up from that point:

see STAT 6610 notes

Rearranging to avoid summing the infinite tail of the distribution...

Converting to C code...

```
#include <math.h>
double AttackerSuccessProbability(double q, int z)
{
    double p = 1.0 - q;     double lambda =
z * (q / p);     double sum = 1.0;     int
i, k;
    for (k = 0; k <= z; k++)
    { double poisson = exp(-lambda); for (i = 1;
        i <= k; i++) poisson *= lambda / i;
        sum -= poisson * (1 - pow(q / p, z - k));
    }     return sum;
}
```

Running some results, we can see the probability drop off exponentially with z.

```
q=0.1 z=0     P=1.0000000 z=1
P=0.2045873 z=2    P=0.0509779
z=3          P=0.0131722   z=4
P=0.0034552 z=5    P=0.0009137
z=6          P=0.0002428   z=7
P=0.0000647 z=8    P=0.0000173
z=9          P=0.0000046   z=10
P=0.0000012

q=0.3 z=0     P=1.0000000 z=5
P=0.1773523 z=10   P=0.0416605
z=15         P=0.0101008   z=20
P=0.0024804 z=25   P=0.0006132
z=30         P=0.0001522   z=35
P=0.0000379 z=40   P=0.0000095
z=45         P=0.0000024   z=50
P=0.0000006
```

Solving for P less than 0.1%...

```
P < 0.001  q=0.10     z=5
q=0.15   z=8 q=0.20   z=11
q=0.25   z=15 q=0.30  z=24
q=0.35   z=41 q=0.40  z=89
q=0.45   z=340
```

12. Conclusion

We have proposed a system for electronic transactions without relying on trust. We started with the usual framework of coins made from digital signatures, which provides strong control of ownership, but is incomplete without a way to prevent double-spending. To solve this, we proposed a peer-to-peer network using proof-of-work to record a public history of transactions that quickly becomes computationally impractical for an attacker to change if honest nodes control a majority of CPU power. The network is robust in its unstructured simplicity. Nodes work all at once with little coordination. They do not need to be identified, since messages are not routed to any particular place and only need to be delivered on a best effort basis. Nodes can leave and rejoin the network at will, accepting the proof-of-work chain as proof of what happened while they were gone. They vote with their CPU power, expressing their acceptance of valid blocks by working on extending them and rejecting invalid blocks by refusing to work on them. Any needed rules and incentives can be enforced with this consensus mechanism.

References

[1] W. Dai, "b-money," http://www.weidai.com/bmoney.txt, 1998.

[2] H. Massias, X.S. Avila, and J.-J. Quisquater, "Design of a secure timestamping service with minimal trust requirements," In *20th Symposium on Information Theory in the Benelux*, May 1999.

[3] S. Haber, W.S. Stornetta, "How to time-stamp a digital document," In *Journal of Cryptology*, vol 3, no 2, pages 99-111, 1991.

[4] D. Bayer, S. Haber, W.S. Stornetta, "Improving the efficiency and reliability of digital time-stamping,"

Side 8

In *Sequences II: Methods in Communication, Security and Computer Science*, pages 329-334, 1993.

[5] S. Haber, W.S. Stornetta, "Secure names for bit-strings," In *Proceedings of the 4th ACM Conference on Computer and Communications Security*, pages 28-35, April 1997.

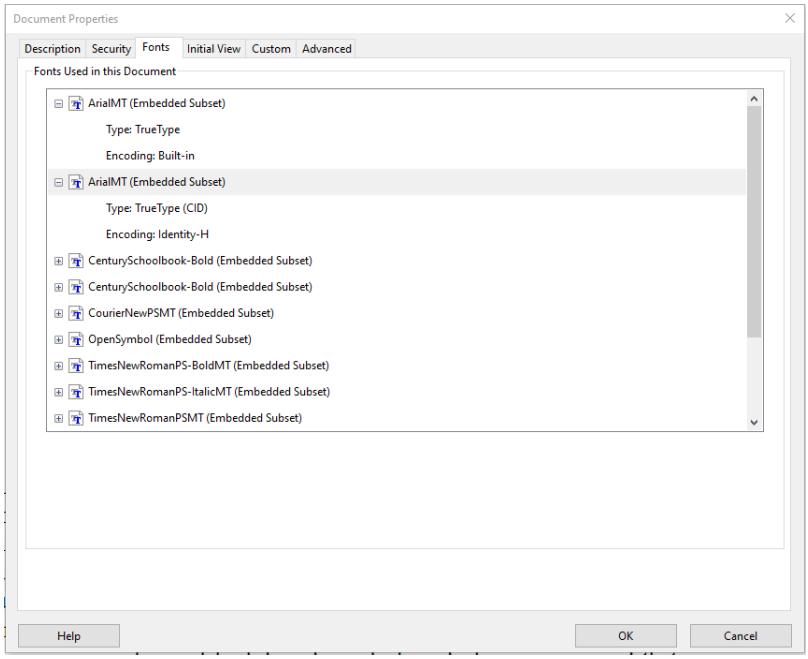
[6] A. Back, "Hashcash - a denial of service counter-measure," http://www.hashcash.org/papers/hashcash.pdf, 2002.

[7] R.C. Merkle, "Protocols for public key cryptosystems," In *Proc. 1980 Symposium on Security and Privacy*, IEEE Computer Society, pages 122-133, April 1980.

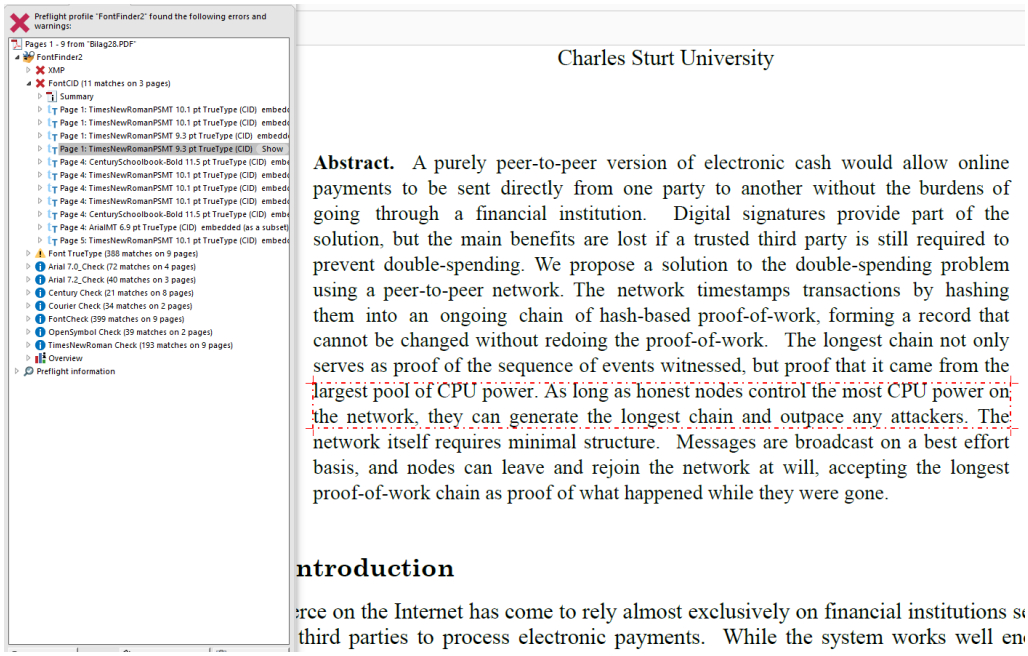
[8] W. Feller, "An introduction to probability theory and its applications," 1957.

4.21 Vedlegg 21: Innholdsanalyse «Bilag 28.pdf»

4.21.1 Fontoversikt

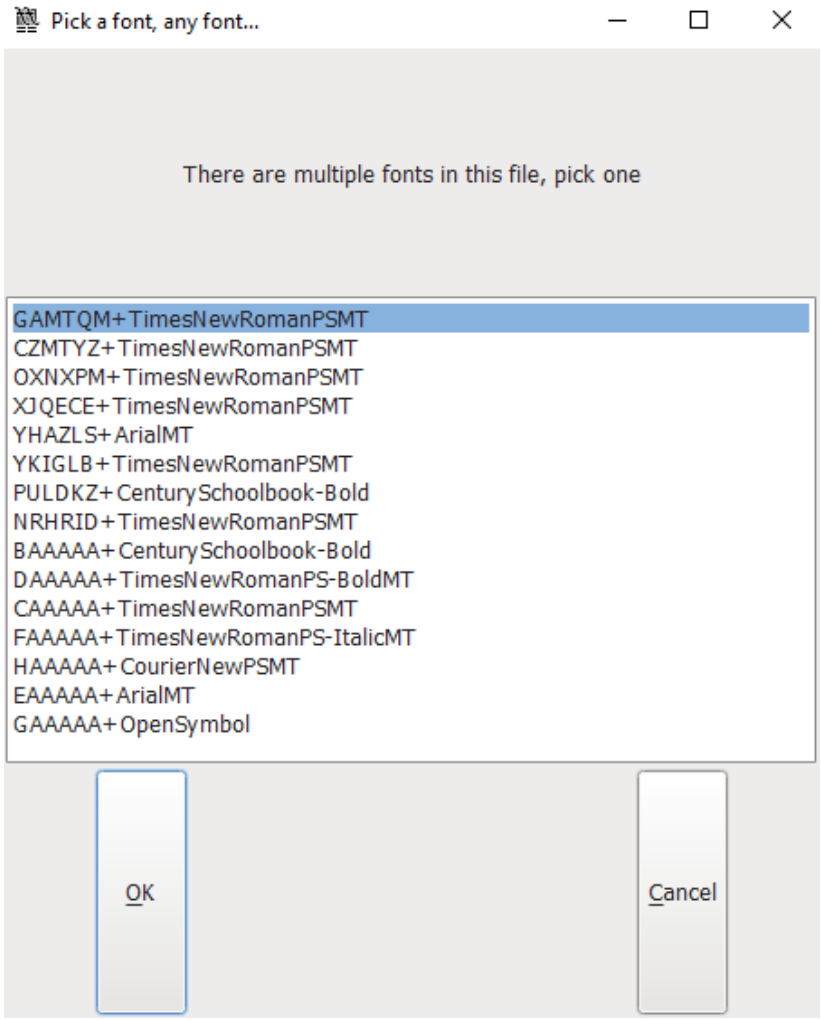


4.21.2 Font-type



4.21.3 Font-metadata fra FontForge

Font-metadata-skjermbildene for «Bilag 28.pdf» er eksakt lik for alle fonter som font-metadataene funnet i «SSRN–id3440802», med unntak av Helvetica-fonten. Se derfor bilagene i vedlegg *Vedlegg 16: Innholdsanalyse «SSRN-id3440802.pdf»* for detaljer.

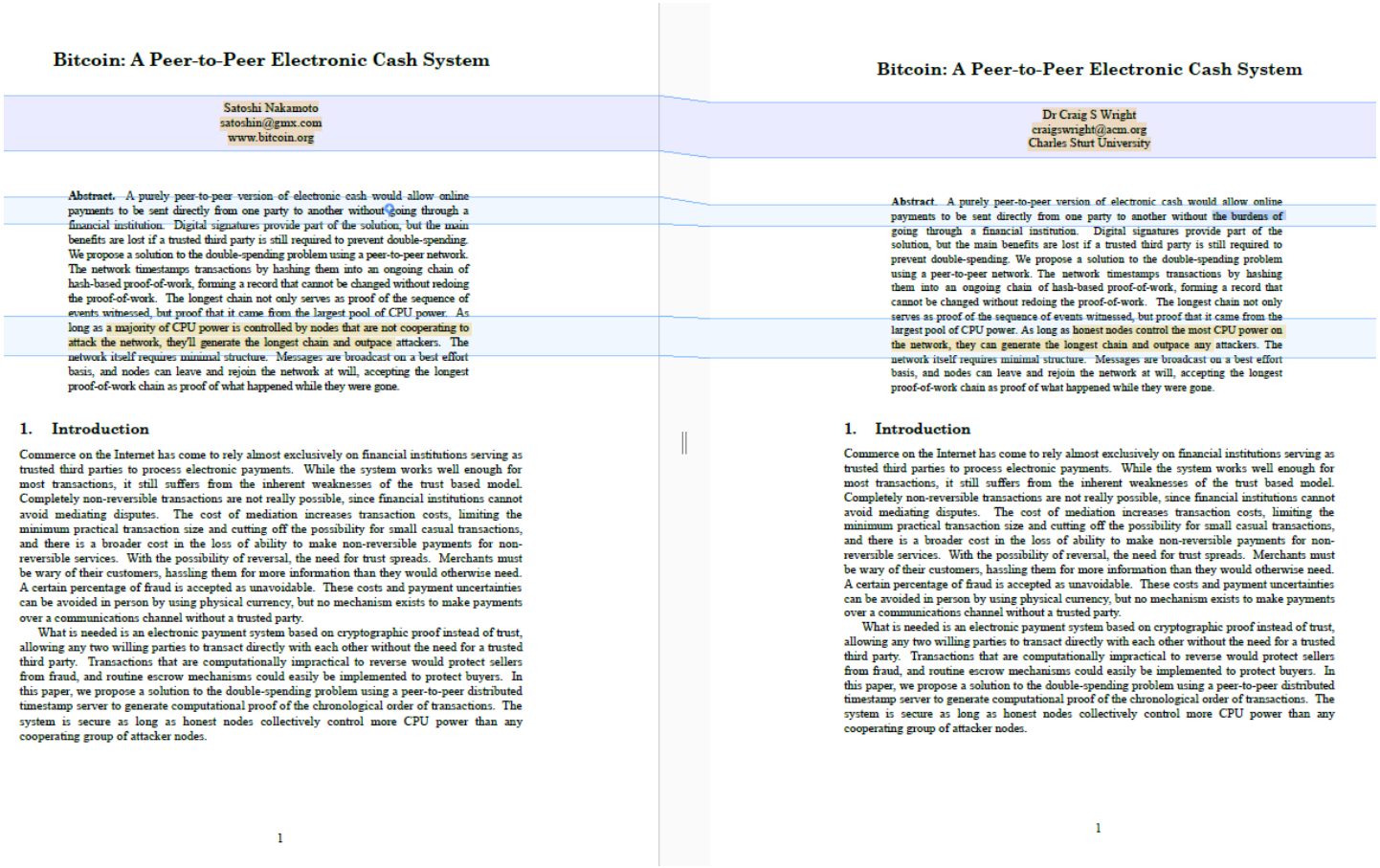


1346

4.21.4 Tekstforskjeller fra «bitcoin.pdf»

All «endret» tekst er uthevet i gult, all tillegg av tekst er uthevet i blått og andre endringer er markert med et hvitt kryss på rød bakgrunn.

Side 1:



Bitcoin: A Peer-to-Peer Electronic Cash System

Satoshi Nakamoto
satoshin@gmx.com
www.bitcoin.org

Abstract. A purely peer-to-peer version of electronic cash would allow online payments to be sent directly from one party to another without going through a financial institution. Digital signatures provide part of the solution, but the main benefits are lost if a trusted third party is still required to prevent double-spending. We propose a solution to the double-spending problem using a peer-to-peer network. The network timestamps transactions by hashing them into an ongoing chain of hash-based proof-of-work, forming a record that cannot be changed without redoing the proof-of-work. The longest chain not only serves as proof of the sequence of events witnessed, but proof that it came from the largest pool of CPU power. As long as a majority of CPU power is controlled by nodes that are not cooperating to attack the network, they'll generate the longest chain and outpace attackers. The network itself requires minimal structure. Messages are broadcast on a best effort basis, and nodes can leave and rejoin the network at will, accepting the longest proof-of-work chain as proof of what happened while they were gone.

1. Introduction

Commerce on the Internet has come to rely almost exclusively on financial institutions serving as trusted third parties to process electronic payments. While the system works well enough for most transactions, it still suffers from the inherent weaknesses of the trust based model. Completely non-reversible transactions are not really possible, since financial institutions cannot avoid mediating disputes. The cost of mediation increases transaction costs, limiting the minimum practical transaction size and cutting off the possibility for small casual transactions, and there is a broader cost in the loss of ability to make non-reversible payments for non-reversible services. With the possibility of reversal, the need for trust spreads. Merchants must be wary of their customers, hassling them for more information than they would otherwise need. A certain percentage of fraud is accepted as unavoidable. These costs and payment uncertainties can be avoided in person by using physical currency, but no mechanism exists to make payments over a communications channel without a trusted party.

What is needed is an electronic payment system based on cryptographic proof instead of trust, allowing any two willing parties to transact directly with each other without the need for a trusted third party. Transactions that are computationally impractical to reverse would protect sellers from fraud, and routine escrow mechanisms could easily be implemented to protect buyers. In this paper, we propose a solution to the double-spending problem using a peer-to-peer distributed timestamp server to generate computational proof of the chronological order of transactions. The system is secure as long as honest nodes collectively control more CPU power than any cooperating group of attacker nodes.

1

Bitcoin: A Peer-to-Peer Electronic Cash System

Dr Craig S Wright
craigswright@acm.org
Charles Sturt University

Abstract. A purely peer-to-peer version of electronic cash would allow online payments to be sent directly from one party to another without the burdens of going through a financial institution. Digital signatures provide part of the solution, but the main benefits are lost if a trusted third party is still required to prevent double-spending. We propose a solution to the double-spending problem using a peer-to-peer network. The network timestamps transactions by hashing them into an ongoing chain of hash-based proof-of-work, forming a record that cannot be changed without redoing the proof-of-work. The longest chain not only serves as proof of the sequence of events witnessed, but proof that it came from the largest pool of CPU power. As long as honest nodes control the most CPU power on the network, they can generate the longest chain and outpace any attackers. The network itself requires minimal structure. Messages are broadcast on a best effort basis, and nodes can leave and rejoin the network at will, accepting the longest proof-of-work chain as proof of what happened while they were gone.

1. Introduction

Commerce on the Internet has come to rely almost exclusively on financial institutions serving as trusted third parties to process electronic payments. While the system works well enough for most transactions, it still suffers from the inherent weaknesses of the trust based model. Completely non-reversible transactions are not really possible, since financial institutions cannot avoid mediating disputes. The cost of mediation increases transaction costs, limiting the minimum practical transaction size and cutting off the possibility for small casual transactions, and there is a broader cost in the loss of ability to make non-reversible payments for non-reversible services. With the possibility of reversal, the need for trust spreads. Merchants must be wary of their customers, hassling them for more information than they would otherwise need. A certain percentage of fraud is accepted as unavoidable. These costs and payment uncertainties can be avoided in person by using physical currency, but no mechanism exists to make payments over a communications channel without a trusted party.

What is needed is an electronic payment system based on cryptographic proof instead of trust, allowing any two willing parties to transact directly with each other without the need for a trusted third party. Transactions that are computationally impractical to reverse would protect sellers from fraud, and routine escrow mechanisms could easily be implemented to protect buyers. In this paper, we propose a solution to the double-spending problem using a peer-to-peer distributed timestamp server to generate computational proof of the chronological order of transactions. The system is secure as long as honest nodes collectively control more CPU power than any cooperating group of attacker nodes.

1

1346

1347

Side 2:

Ingen forskjeller

Side 3:

Ingen forskjeller

1347

1348

Side 4:

New transaction broadcasts do not necessarily need to reach all nodes. As long as they reach many nodes, they will get into a block before long. Block broadcasts are also tolerant of dropped messages. If a node does not receive a block, it will request it when it receives the next block and realizes it missed one.

6. Incentive

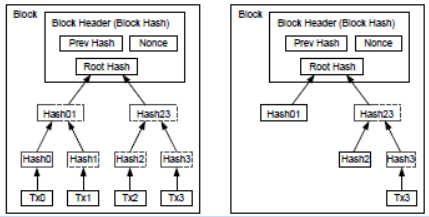
By convention, the first transaction in a block is a special transaction that starts a new coin owned by the creator of the block. This adds an incentive for nodes to support the network, and provides a way to initially distribute coins into circulation, since there is no central authority to issue them. The steady addition of a constant of amount of new coins is analogous to gold miners expending resources to add gold to circulation. In our case, it is CPU time and electricity that is expended.

The incentive can also be funded with transaction fees. If the output value of a transaction is less than its input value, the difference is a transaction fee that is added to the incentive value of the block containing the transaction. Once a predetermined number of coins have entered circulation, the incentive can transition entirely to transaction fees and be completely inflation free.

The incentive may help encourage nodes to stay honest. If a greedy attacker is able to assemble more CPU power than all the honest nodes, he would have to choose between using it to defraud people by stealing back his payments, or using it to generate new coins. He ought to find it more profitable to play by the rules, such rules that favour him with more new coins than everyone else combined, than to undermine the system and the validity of his own wealth.

7. Reclaiming Disk Space

Once the latest transaction in a coin is buried under enough blocks, the spent transactions before it can be discarded to save disk space. To facilitate this without breaking the block's hash, transactions are hashed in a Merkle Tree [7][2][5], with only the root included in the block's hash. Old blocks can then be compacted by stubbing off branches of the tree. The interior hashes do not need to be stored.



A block header with no transactions would be about 80 bytes. If we suppose blocks are generated every 10 minutes, 80 bytes * 6 * 24 * 365 = 4.2MB per year. With computer systems typically selling with 2GB of RAM as of 2008, and Moore's Law predicting current growth of 1.2GB per year, storage should not be a problem even if the block headers must be kept in memory.

4

New transaction broadcasts do not necessarily need to reach all nodes. As long as they reach many nodes, they will get into a block before long. Block broadcasts are also tolerant of dropped messages. If a node does not receive a block, it will request it when it receives the next block and realizes it missed one.

6. Incentive

By convention, the first transaction in a block is a special transaction that starts a new coin owned by the creator of the block. This adds an incentive for nodes to support the network, and provides a way to initially distribute coins into circulation, since there is no central authority to issue them. The steady addition of a constant of amount of new coins is analogous to gold miners expending resources to add gold to circulation. In our case, it is CPU time and electricity that is expended.

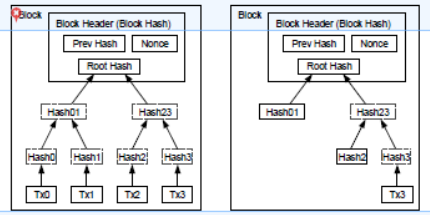
The incentive can also be funded with transaction fees. If the output value of a transaction is less than its input value, the difference is a transaction fee that is added to the incentive value of the block containing the transaction. Once a predetermined number of coins have entered circulation, the incentive can transition entirely to transaction fees and be completely inflation free.

The incentive may help encourage nodes to stay honest. If a greedy attacker is able to assemble more CPU power than all the honest nodes, he would have to choose between using it to defraud people by stealing back his payments, or using it to generate new coins. He ought to find it more profitable to play by the rules, such rules that favour him with more new coins than everyone else combined, than to undermine the system and the validity of his own wealth.

To compensate for increasing hardware speed and varying interest in running nodes over time, the proof-of-work difficulty is determined by a moving average targeting an average number of blocks per hour. If they're generated too fast, the difficulty increases.

7. Reclaiming Disk Space

Once the latest transaction in a coin is buried under enough blocks, the spent transactions before it can be discarded to save disk space. To facilitate this without breaking the block's hash, transactions are hashed in a Merkle Tree [7][2][5], with only the root included in the block's hash. Old blocks can then be compacted by stubbing off branches of the tree. The interior hashes do not need to be stored.



A block header with no transactions would be about 80 bytes. If we suppose blocks are generated every 10 minutes, 80 bytes * 6 * 24 * 365 = 4.2MB per year. With computer systems typically selling with 2GB of RAM as of 2008, and Moore's Law predicting current growth of 1.2GB per year, storage should not be a problem even if the block headers must be kept in memory.

4

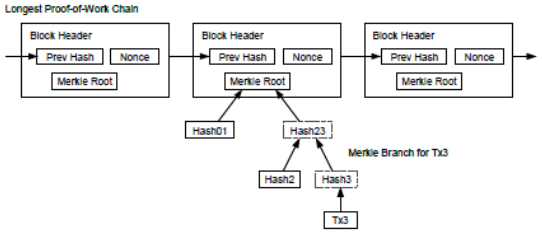
1348

1349

Side 5:

8. Simplified Payment Verification

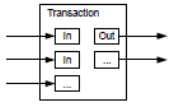
It is possible to verify payments without running a full network node. A user only needs to keep a copy of the block headers of the longest proof-of-work chain, which he can get by querying network nodes until he's convinced he has the longest chain, and obtain the Merkle branch linking the transaction to the block it's timestamped in. He can't check the transaction for himself, but by linking it to a place in the chain, he can see that a network node has accepted it, and blocks added after it further confirm the network has accepted it.



As such, the verification is reliable as long as honest nodes control the network, but is more vulnerable if the network is overpowered by an attacker. While network nodes can verify transactions for themselves, the simplified method can be fooled by an attacker's fabricated transactions for as long as the attacker can continue to overpower the network. One strategy to protect against this would be to accept alerts from network nodes when they detect an invalid block, prompting the user's software to download the full block and alerted transactions to confirm the inconsistency. Businesses that receive frequent payments will probably still want to run their own nodes for more independent security and quicker verification.

9. Combining and Splitting Value

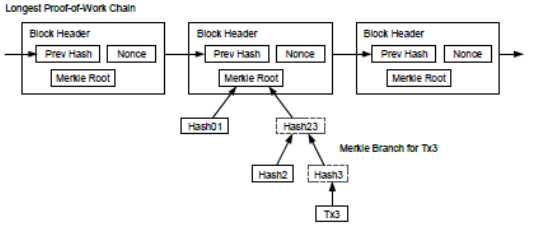
Although it would be possible to handle coins individually, it would be unwieldy to make a separate transaction for every cent in a transfer. To allow value to be split and combined, transactions contain multiple inputs and outputs. Normally there will be either a single input from a larger previous transaction or multiple inputs combining smaller amounts, and at most two outputs: one for the payment, and one returning the change, if any, back to the sender.



It should be noted that fan-out, where a transaction depends on several transactions, and those transactions depend on many more, is not a problem here. There is never the need to extract a complete standalone copy of a transaction's history.

8. Simplified Payment Verification

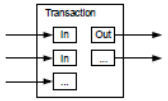
It is possible to verify payments without running a full network node. A user only needs to keep a copy of the block headers of the longest proof-of-work chain, which he can get by querying network nodes until he's convinced he has the longest chain, and obtain the Merkle branch linking the transaction to the block it's timestamped in. He can't check the transaction for himself, but by linking it to a place in the chain, he can see that a network node has accepted it, and blocks added after it further confirm the network has accepted it.



As such, the verification is reliable as long as honest nodes control the network, but is more vulnerable if the network is overpowered by an attacker. While network nodes can verify transactions for themselves and are only vulnerable to reversal, the simplified method can be fooled by an attacker's fabricated transactions for as long as the attacker can continue to overpower the network. One strategy to protect against this would be to accept alerts from network nodes when they detect an invalid block, prompting the user's software to download the full block and alerted transactions to confirm the inconsistency. Businesses that receive frequent payments will probably still want to run their own nodes for more independent security and quicker verification.

9. Combining and Splitting Value

Although it would be possible to handle coins individually, it would be unwieldy to make a separate transaction for every cent in a transfer. To allow value to be split and combined, transactions contain multiple inputs and outputs. Normally there will be either a single input from a larger previous transaction or multiple inputs combining smaller amounts, and at most two outputs: one for the payment, and one returning the change, if any, back to the sender.



It should be noted that fan-out, where a transaction depends on several transactions, and those transactions depend on many more, is not a problem here. There is never the need to extract a complete standalone copy of a transaction's history.

5

1349

1350

Side 6:

Ingen forskjeller

1350

1351

Side 7:

Given our assumption that $p > q$, the probability drops exponentially as the number of blocks the attacker has to catch up with increases. With the odds against him, if he doesn't make a lucky lunge forward early on, his chances become vanishingly small as he falls further behind.

We now consider how long the recipient of a new transaction needs to wait before being sufficiently certain the sender can't change the transaction. We assume the sender is an attacker who wants to make the recipient believe he paid him for a while, then switch it to pay back to himself after some time has passed. The receiver will be alerted when that happens, but the sender hopes it will be too late.

The receiver generates a new key pair and gives the public key to the sender shortly before signing. This prevents the sender from preparing a chain of blocks ahead of time by working on it continuously until he is lucky enough to get far enough ahead, then executing the transaction at that moment. Once the transaction is sent, the dishonest sender starts working in secret on a parallel chain containing an alternate version of his transaction.

The recipient waits until the transaction has been added to a block and z blocks have been linked after it. He doesn't know the exact amount of progress the attacker has made, but assuming the honest blocks took the average expected time per block, the attacker's potential progress will be a Poisson distribution with expected value:

$$\lambda = z\frac{q}{p}$$

To get the probability the attacker could still catch up now, we multiply the Poisson density for each amount of progress he could have made by the probability he could catch up from that point:

$$\sum_{k=0}^{\infty} \frac{\lambda^k e^{-\lambda}}{k!} \cdot \begin{cases} (q/p)^{(z-k)} & \text{if } k \le z \\ 1 & \text{if } k > z \end{cases}$$

Rearranging to avoid summing the infinite tail of the distribution...

$$1 - \sum_{k=0}^{z} \frac{\lambda^k e^{-\lambda}}{k!} \left(1 - (q/p)^{(z-k)}\right)$$

Converting to C code...

```
#include <math.h>
double AttackerSuccessProbability(double q, int z)
{
    double p = 1.0 - q;
    double lambda = z * (q / p);
    double sum = 1.0;
    int i, k;
    for (k = 0; k <= z; k++)
    {
        double poisson = exp(-lambda);
        for (i = 1; i <= k; i++)
            poisson *= lambda / i;
        sum -= poisson * (1 - pow(q / p, z - k));
    }
    return sum;
}
```

Given our assumption that $p > q$, the probability drops exponentially as the number of blocks the attacker has to catch up with increases. With the odds against him, if he doesn't make a lucky lunge forward early on, his chances become vanishingly small as he falls further behind.

We now consider how long the recipient of a new transaction needs to wait before being sufficiently certain the sender can't change the transaction. We assume the sender is an attacker who wants to make the recipient believe he paid him for a while, then switch it to pay back to himself after some time has passed. The receiver will be alerted when that happens, but the sender hopes it will be too late.

The receiver generates a new key pair and gives the public key to the sender shortly before signing. This prevents the sender from preparing a chain of blocks ahead of time by working on it continuously until he is lucky enough to get far enough ahead, then executing the transaction at that moment. Once the transaction is sent, the dishonest sender starts working in secret on a parallel chain containing an alternate version of his transaction.

The recipient waits until the transaction has been added to a block and z blocks have been linked after it. He doesn't know the exact amount of progress the attacker has made, but assuming the honest blocks took the average expected time per block, the attacker's potential progress will be a Poisson distribution with expected value:

$$\lambda = z\frac{q}{p}$$

To get the probability the attacker could still catch up now, we multiply the Poisson density for each amount of progress he could have made by the probability he could catch up from that point:

$$\sum_{k=0}^{\infty} \frac{\lambda^k e^{-\lambda}}{k!} \cdot \begin{cases} (q/p)^{(z-k)} & \text{if } k \le z \\ 1 & \text{if } k > z \end{cases}$$

Rearranging to avoid summing the infinite tail of the distribution...

$$1 - \sum_{k=0}^{z} \frac{\lambda^k e^{-\lambda}}{k!} \left(1 - (q/p)^{(z-k)}\right)$$

Converting to C code...

```
#include <math.h>
double AttackerSuccessProbability(double q, int z)
{
    double p = 1.0 - q;
    double lambda = z * (q / p);
    double sum = 1.0;
    int i, k;
    for (k = 0; k <= z; k++)
    {
        double poisson = exp(-lambda);
        for (i = 1; i <= k; i++)
            poisson *= lambda / i;
        sum -= poisson * (1 - pow(q / p, z - k));
    }
    return sum;
}
```


7

1351

1352

Side 8:

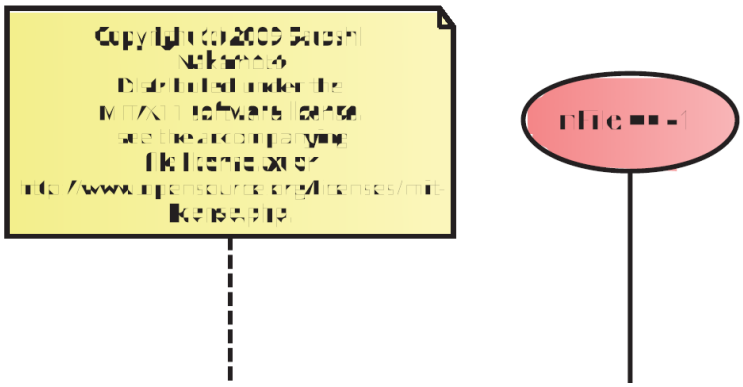
Ingen forskjeller

Side 9:

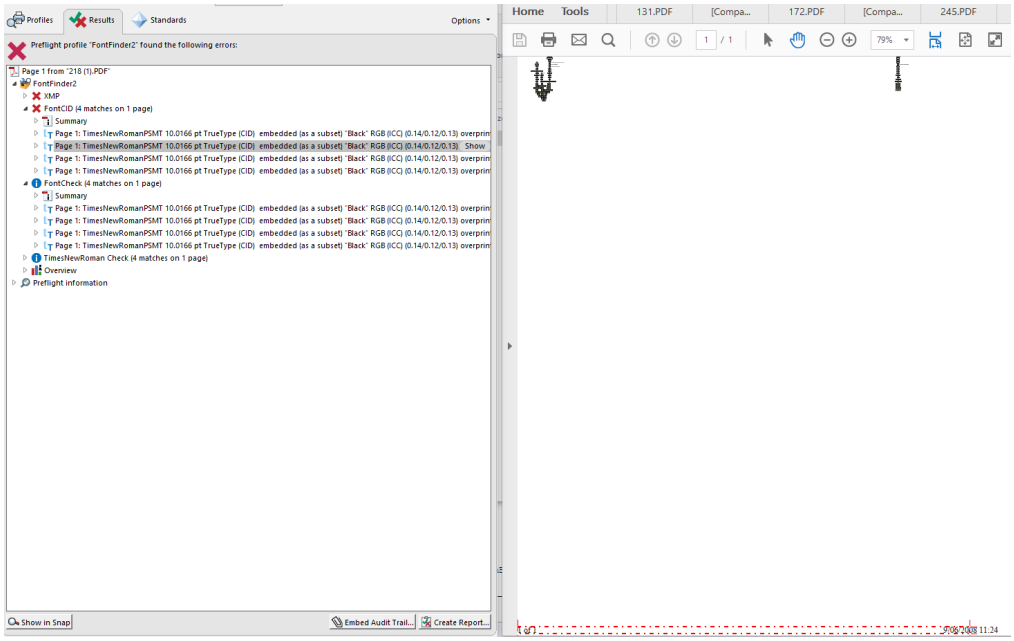
Ingen forskjeller

4.22 Vedlegg 22: Innholdsanalyse «Bilag 29.pdf»

4.22.1 Copyright i header

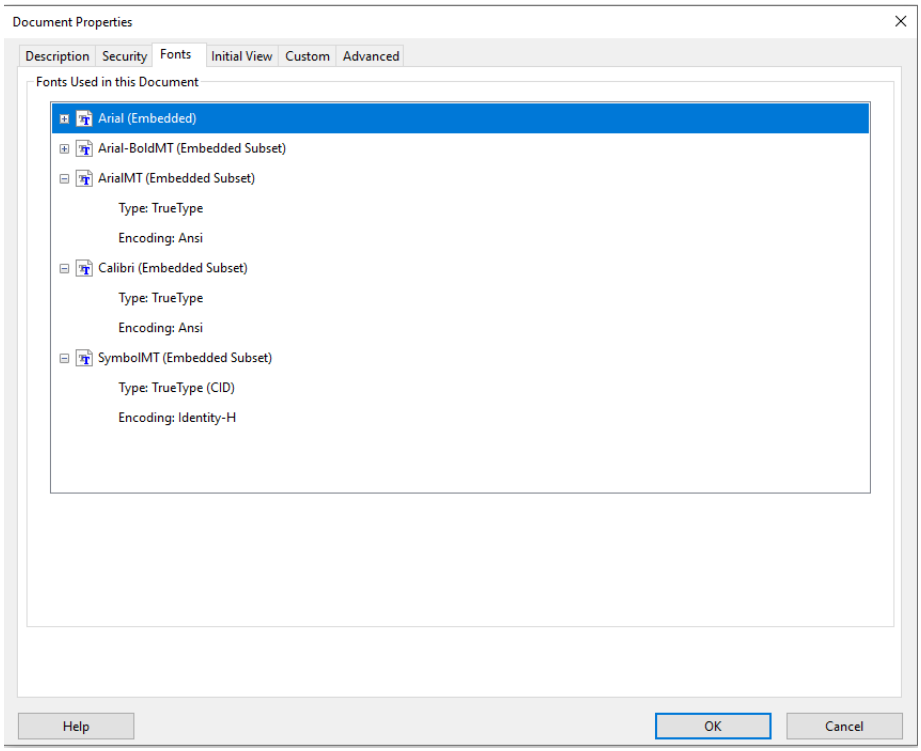


4.22.2 Font-type

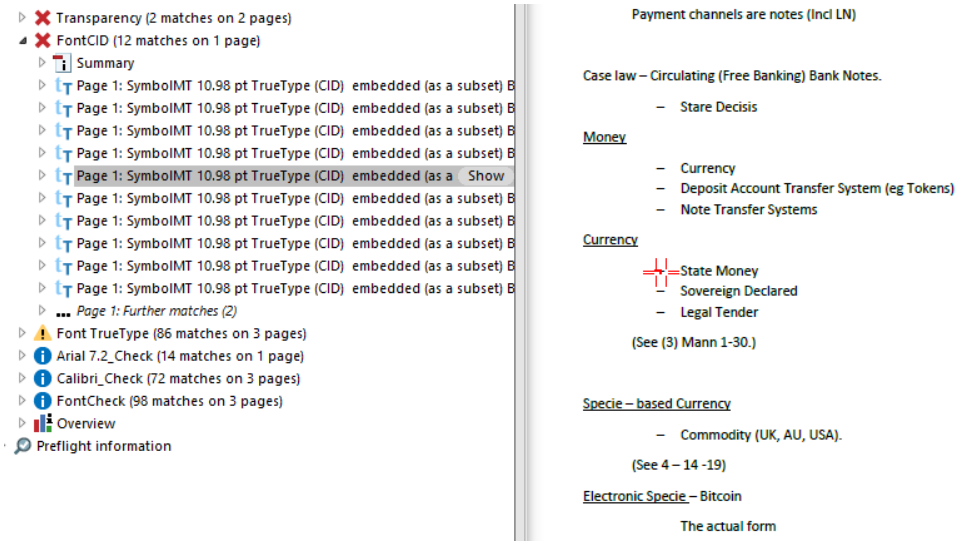


4.23 Vedlegg 23: Innholdsanalyse «Bilag 30.doc»

4.23.1 Fontoversikt



4.23.2 Font-type

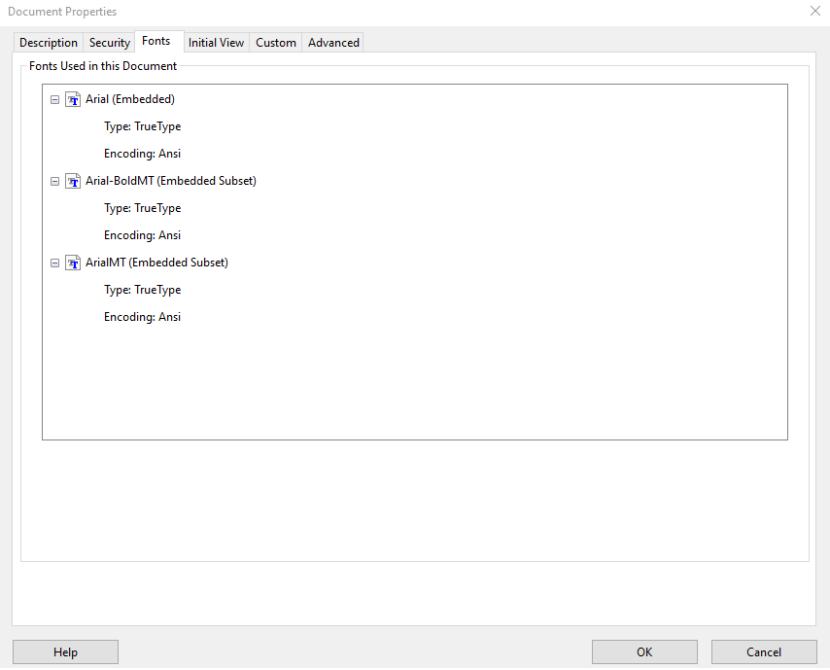


4.24 Vedlegg 24: Font-oversikt «Bilag 30.doc» og «Bilag 31.doc»

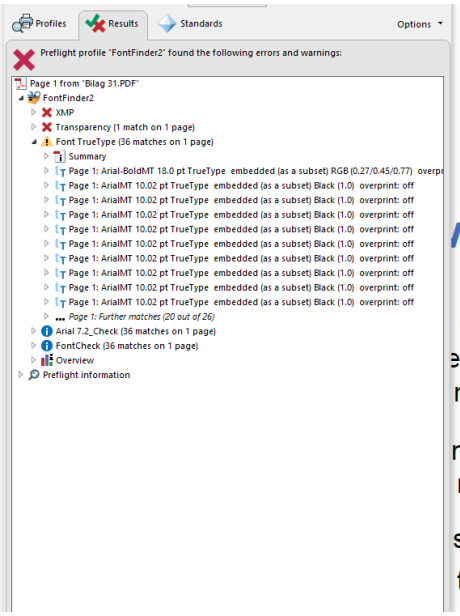
Bilag 30.doc og Bilag 31.doc: Font-oversikt		
Font-familie	**«Bilag 30.doc»**	**«Bilag 31.doc»**
Arial	**Arial** Type: TrueType Encoding: Ansi	**Arial** Type: TrueType Encoding: Ansi
	Arial-BoldMT Type: TrueType Encoding: Ansi	**Arial-BoldMT** Type: TrueType Encoding: Ansi
	Arial-MT Type: TrueType Encoding: Ansi	**Arial-MT** Type: TrueType Encoding: Ansi
Calibri	**Calibri** Type: TrueType Encoding: Ansi	
Symbol	**SymbolMT** Type: TrueType (CID) Encoding: Identity-H	

4.25 Vedlegg 25: Innholdsanalyse «Bilag 31.doc»

4.25.1 Fontoversikt



4.25.2 Font-type



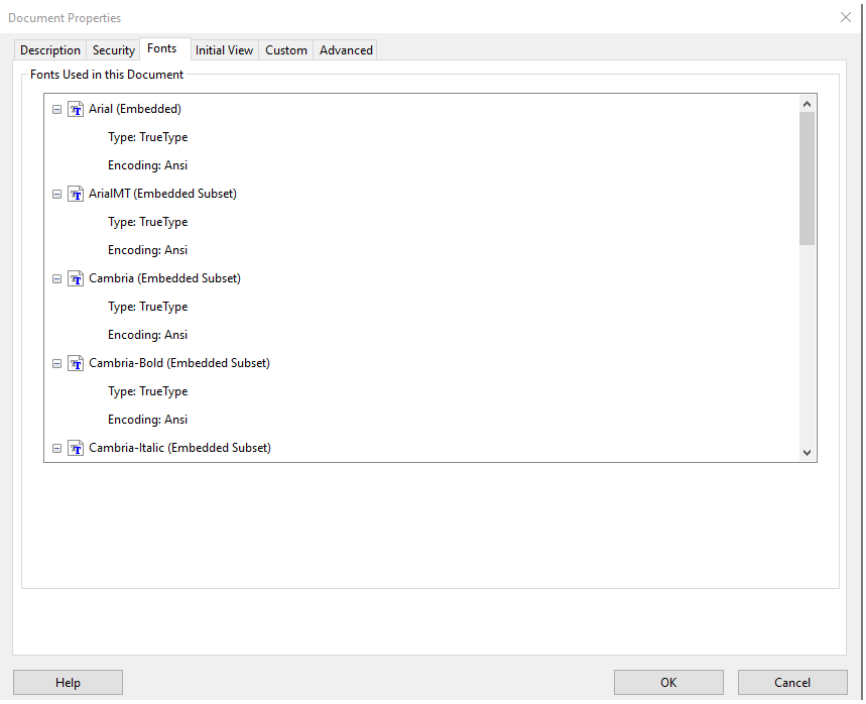
recipient B faces minimal, if any, ongoing
e remains an ever diminishing risk of a bloc
multiple blocks, this becomes a near zero

nsaction, the risk to B is minimal as long a
more certain).

sion from (A) and a miner, and this is prob
the simple addition of a tx into the memp

4.26 Vedlegg 26: Innholdsanalyse «Bilag 32.doc»

4.26.1 Font-type og encoding



4.26.2 Forskjeller fra «bitcoin.pdf»

Her vises kun «Bilag 32.doc» i bilagets PDF-format. All «endret» tekst er uthevet i gult, all tillegg av tekst er uthevet i blått og andre endringer er markert med et hvitt kryss på rød bakgrunn.

Side 1:

Bitcoin: A Peer-to-Peer Electronic Cash System

Abstract. A purely peer-to-peer version of electronic cash would allow online payments to be sent directly from one party to another without going through a financial institution. Digital signatures provide part of the solution, but the main benefits are lost if a trusted third party is still required to prevent double-spending. We propose a solution to the double-spending problem using a peer-to-peer network. The network timestamps transactions by hashing them into an ongoing chain of hash-based proof-of-work, forming a record that cannot be changed without redoing the proof-of-work. The longest chain not only serves as proof of the sequence of events witnessed, but proof that it came from the largest pool of CPU power. As long as a majority of CPU power is controlled by nodes that are not cooperating to attack the network, they'll generate the longest chain and outpace attackers. The network itself requires minimal structure. Messages are broadcast on a best effort basis, and nodes can leave and rejoin the network at will, accepting the longest proof-of-work chain as proof of what happened while they were gone.

1. Introduction

Commerce on the Internet has come to rely almost exclusively on financial institutions serving as trusted third parties to process electronic payments. While the system works well enough for most transactions, it still suffers from the inherent weaknesses of the trust based model. Completely non-reversible transactions are not really possible, since financial institutions cannot avoid mediating disputes. The cost of mediation increases transaction costs, limiting the minimum practical transaction size and cutting off the possibility for small casual transactions, and there is a broader cost in the loss of ability to make non-reversible payments for non- reversible services. With the possibility of reversal, the need for trust spreads. Merchants must be wary of their customers, hassling them for more information than they would otherwise need. A certain percentage of fraud is accepted as unavoidable. These costs and payment uncertainties can be avoided in person by using physical currency, but no mechanism exists to make payments over a communications channel without a trusted party.

What is needed is an electronic payment system based on cryptographic proof instead of trust, allowing any two willing parties to transact directly with each other without the need for a trusted third party. Transactions that are computationally impractical to reverse would protect sellers from fraud, and routine escrow mechanisms could easily be implemented to protect buyers. In this paper, we propose a solution to the double-spending problem using a peer-to-peer distributed timestamp server to generate computational proof of the chronological order of transactions. The system is secure as long as honest nodes collectively control more CPU power than any cooperating group of attacker nodes.

2. Transactions

Side 2:

We define an electronic coin as a chain of digital signatures. Each owner transfers the coin to the next by digitally signing a hash of the previous transaction and the public key of the next owner and adding these to the end of the coin. A payee can verify the signatures to verify the chain of ownership.

The problem of course is the payee can't verify that one of the owners did not double-spend the coin. A common solution is to introduce a trusted central authority, or mint, that checks every transaction for double spending. After each transaction, the coin must be returned to the mint to issue a new coin, and only coins issued directly from the mint are trusted not to be double-spent. The problem with this solution is that the fate of the entire money system depends on the company running the mint, with every transaction having to go through them, just like a bank.

We need a way for the payee to know that the previous owners did not sign any earlier transactions. For our purposes, the earliest transaction is the one that counts, so we don't care about later attempts to double-spend. The only way to confirm the absence of a transaction is to be aware of all transactions. In the mint based model, the mint was aware of all transactions and decided which arrived first. To accomplish this without a trusted party, transactions must be publicly announced [1], and we need a system for participants to agree on a single history of the order in which they were received. The payee needs proof that at the time of each transaction, the majority of nodes agreed it was the first received.

3. Timestamp Server

The solution we propose begins with a timestamp server. A timestamp server works by taking a hash of a block of items to be timestamped and widely publishing the hash, such as in a newspaper or Usenet post [2-5]. The timestamp proves that the data must have existed at the time, obviously, in order to get into the hash. Each timestamp includes the previous timestamp in its hash, forming a chain, with each additional timestamp reinforcing the ones before it.

4. Proof-of-Work

To implement a distributed timestamp server on a peer-to-peer basis, we will need to use a proof- of-work system similar to Adam Back's Hashcash [6], rather than newspaper or Usenet posts. The proof-of-work involves scanning for a value that when hashed, such as with SHA-256, the hash begins with a number of zero bits. The average work required is exponential in the number of zero bits required and can be verified by executing a single hash.

For our timestamp network, we implement the proof-of-work by incrementing a nonce in the block until a value is found that gives the block's hash the required zero bits. Once the CPU effort has been expended to make it satisfy the proof-of-work, the block cannot be changed without redoing the work. As later blocks are chained after it, the work to change the block would include redoing all the blocks after it.

The proof-of-work also solves the problem of determining representation in majority decision making. If the majority were based on one-IP-address-one-vote, it could be subverted by anyone able to allocate many IPs. Proof-of-work is essentially one-CPU-one-vote. The majority decision is represented by the longest chain, which has the greatest proof-of-work effort invested in it. If a majority of CPU power is controlled by honest nodes, the honest chain will grow the fastest and outpace any competing chains. To modify a

Side 3:

past block, an attacker would have to redo the proof-of-work of the block and all blocks after it and then catch up with and surpass the work of the honest nodes. We will show later that the probability of a slower attacker catching up diminishes exponentially as subsequent blocks are added.
To compensate for increasing hardware speed and varying interest in running nodes over time, the proof-of-work difficulty is determined by a moving average targeting an average number of blocks per hour. If they're generated too fast, the difficulty increases.

5. Network

The steps to run the network are as follows:

1) New transactions are broadcast to all nodes.
2) Each node collects new transactions into a block.
3) Each node works on finding a difficult proof-of-work for its block.
4) When a node finds a proof-of-work, it broadcasts the block to all nodes.
5) Nodes accept the block only if all transactions in it are valid and not already spent.
6) Nodes express their acceptance of the block by working on creating the next block in the chain, using the hash of the accepted block as the previous hash.

Nodes always consider the longest chain to be the correct one and will keep working on extending it. If two nodes broadcast different versions of the next block simultaneously, some nodes may receive one or the other first. In that case, they work on the first one they received, but save the other branch in case it becomes longer. The tie will be broken when the next proof- of-work is found and one branch becomes longer; the nodes that were working on the other branch will then switch to the longer one.

New transaction broadcasts do not necessarily need to reach all nodes. As long as they reach many nodes, they will get into a block before long. Block broadcasts are also tolerant of dropped messages. If a node does not receive a block, it will request it when it receives the next block and realizes it missed one.

6. Incentive

By convention, the first transaction in a block is a special transaction that starts a new coin owned by the creator of the block. This adds an incentive for nodes to support the network, and provides a way to initially distribute coins into circulation, since there is no central authority to issue them. The steady addition of a constant of amount of new coins is analogous to gold miners expending resources to add gold to circulation. In our case, it is CPU time and electricity that is expended.
The incentive can also be funded with transaction fees. If the output value of a transaction is less than its input value, the difference is a transaction fee that is added to the incentive value of the block containing the transaction. Once a predetermined number of coins have entered circulation, the incentive can transition entirely to transaction fees and be completely inflation free.
The incentive may help encourage nodes to stay honest. If a greedy attacker is able to assemble more CPU power than all the honest nodes, he would have to choose between using it to defraud people by stealing back his payments, or using it to generate new coins. He ought to find it more profitable to play by the rules, such rules that favour him with more new coins than everyone else combined, than to undermine the system and the validity of his own wealth.

7. Reclaiming Disk Space

Once the latest transaction in a coin is buried under enough blocks, the spent transactions before it can be discarded to save disk space. To facilitate this without breaking the block's hash, transactions are hashed in a Binary Tree [7][2][5], with only the root included in the block's hash. Old blocks can then be compacted by stubbing off branches of the tree. The interior hashes do not need to be stored.

A block header with no transactions would be about 80 bytes. If we suppose blocks are generated every 10 minutes, 80 bytes * 6 * 24 * 365 = 4.2MB per year. With computer systems typically selling with 2GB of

Side 4:

RAM as of 2008, and Moore's Law predicting current growth of 1.2GB per year, storage should not be a problem even if the block headers must be kept in memory.

8. Simplified Payment Verification

It is possible to verify payments without running a full network node. A user only needs to keep a copy of the block headers of the longest proof-of-work chain, which he can get by querying network nodes until he's convinced he has the longest chain, and obtain the branch of the binary tree [7] linking the transaction to the block it's timestamped in. He can't check the transaction for himself, but by linking it to a place in the chain, he can see that a network node has accepted it, and blocks added after it further confirm the network has accepted it.

As such, the verification is reliable as long as honest nodes control the network, but is more vulnerable if the network is overpowered by an attacker. While network nodes can verify transactions for themselves, the simplified method can be fooled by an attacker's fabricated transactions for as long as the attacker can continue to overpower the network. One strategy to protect against this would be to accept alerts from network nodes when they detect an invalid block, prompting the user's software to download the full block and alerted transactions to confirm the inconsistency. Businesses that receive frequent payments will probably still want to run their own nodes for more independent security and quicker verification.

9. Combining and Splitting Value

Although it would be possible to handle coins individually, it would be unwieldy to make a separate transaction for every cent in a transfer. To allow value to be split and combined, transactions contain multiple inputs and outputs. Normally there will be either a single input from a larger previous transaction or multiple inputs combining smaller amounts, and at most two outputs: one for the payment, and one returning the change, if any, back to the sender.

It should be noted that fan-out, where a transaction depends on several transactions, and those transactions depend on many more, is not a problem here. There is never the need to extract a complete standalone copy of a transaction's history.

10. Privacy

The traditional banking model achieves a level of privacy by limiting access to information to the parties involved and the trusted third party. The necessity to announce all transactions publicly precludes this method, but privacy can still be maintained by breaking the flow of information in another place: by keeping public keys anonymous. The public can see that someone is sending an amount to someone else, but without information linking the transaction to anyone. This is similar to the level of information released by stock exchanges, where the time and size of individual trades, the "tape", is made public, but without telling who the parties were.

As an additional firewall, a new key pair should be used for each transaction to keep them from being linked to a common owner. Some linking is still unavoidable with multi-input transactions, which necessarily reveal that their inputs were owned by the same owner. The risk is that if the owner of a key is revealed, linking could reveal other transactions that belonged to the same owner.

11. Calculations

We consider the scenario of an attacker trying to generate an alternate chain faster than the honest chain. Even if this is accomplished, it does not throw the system open to arbitrary changes, such as creating value out of thin air or taking money that never belonged to the attacker. Nodes are not going to accept an invalid transaction as payment, and honest nodes will never accept a block containing them. An attacker can only try to change one of his own transactions to take back money he recently spent.
The race between the honest chain and an attacker chain can be characterized as a Binomial Random Walk.

Side 5:

The success event is the honest chain being extended by one block, increasing its lead by +1, and the failure event is the attacker's chain being extended by one block, reducing the gap by -1.
The probability of an attacker catching up from a given deficit is analogous to a Gambler's Ruin problem. Suppose a gambler with unlimited credit starts at a deficit and plays potentially an infinite number of trials to try to reach breakeven. We can calculate the probability he ever reaches breakeven, or that an attacker ever catches up with the honest chain, as follows [8]:

p = probability an honest node finds the next block
q = probability the attacker finds the next block
q_z = probability the attacker will ever catch up from z blocks behind

Given our assumption that $p > q$, the probability drops exponentially as the number of blocks the attacker has to catch up with increases. With the odds against him, if he doesn't make a lucky lunge forward early on, his chances become vanishingly small as he falls further behind.
We now consider how long the recipient of a new transaction needs to wait before being sufficiently certain the sender can't change the transaction. We assume the sender is an attacker who wants to make the recipient believe he paid him for a while, then switch it to pay back to himself after some time has passed. The receiver will be alerted when that happens, but the sender hopes it will be too late.
The receiver generates a new key pair and gives the public key to the sender shortly before signing. This prevents the sender from preparing a chain of blocks ahead of time by working on it continuously until he is lucky enough to get far enough ahead, then executing the transaction at that moment. Once the transaction is sent, the dishonest sender starts working in secret on a parallel chain containing an alternate version of his transaction.
The recipient waits until the transaction has been added to a block and z blocks have been linked after it. He doesn't know the exact amount of progress the attacker has made, but assuming the honest blocks took the average expected time per block, the attacker's potential progress will be a Poisson distribution with expected value:

To get the probability the attacker could still catch up now, we multiply the Poisson density for each amount of progress he could have made by the probability he could catch up from that point:

Rearranging to avoid summing the infinite tail of the distribution...

Converting to C code...

```
#include <math.h>
double AttackerSuccessProbability(double q, int z)
{
double p = 1.0 - q;
double lambda = z * (q / p); double sum = 1.0;
int i, k;
for (k = 0; k <= z; k++)
```

Side 6:

```
{
double poisson = exp(-lambda); for (i = 1; i <= k; i++)
poisson *= lambda / i;
sum -= poisson * (1 - pow(q / p, z - k));
}
return sum;
}
```

Running some results, we can see the probability drop off exponentially with z.

```
q=0.1
z=0     P=1.000000
z=1     P=0.204587
z=2     P=0.050978
z=3     P=0.013172
z=4     P=0.003455
z=5     P=0.000914
z=6     P=0.000243
z=7     P=0.000065
z=8     P=0.000017
z=9                         P=0.0000046 z=10 P=0.0000012

q=0.3
z=0                         P=1.0000000
z=5 P=0.1773523 z=10 P=0.0416605 z=15 P=0.0101008 z=20 P=0.0024804 z=25 P=0.0006132 z=30
P=0.0001522 z=35 P=0.0000379 z=40 P=0.0000095 z=45 P=0.0000024 z=50   P=0.0000006
```

Solving for P less than 0.1%...

```
P < 0.001 q=0.10              z=5
q=0.15    z=8
q=0.20    z=11
q=0.25    z=15
q=0.30    z=24
q=0.35    z=41
q=0.40    z=89
q=0.45  z=340
```

12. Conclusion

We have proposed a system for electronic transactions without relying on trust. We started with the usual framework of coins made from digital signatures, which provides strong control of ownership, but is incomplete without a way to prevent double-spending. To solve this, we proposed a peer-to-peer network using proof-of-work to record a public history of transactions that quickly becomes computationally impractical for an attacker to change if honest nodes control a majority of CPU power. The network is robust in its unstructured simplicity. Nodes work all at once with little coordination. They do not need to be identified, since messages are not routed to any particular place and only need to be delivered on a best effort basis. Nodes can leave and rejoin the network at will, accepting the proof-of-work chain as proof of what happened while they were gone. They vote with their CPU power, expressing their acceptance of valid blocks by working on extending them and rejecting invalid blocks by refusing to work on them. Any needed rules and incentives can be enforced with this consensus mechanism.

References

[1] W. Dai, "b-money," http://www.weidai.com/bmoney.txt, 1998.

[2] H. Massias, X.S. Avila, and J.-J. Quisquater, "Design of a secure timestamping service with minimal trust requirements," In *20th Symposium on Information Theory in the Benelux*, May 1999.

[3] S. Haber, W.S. Stornetta, "How to time-stamp a digital document," In *Journal of Cryptology*, vol 3, no 2, pages 99-111, 1991.

[4] D. Bayer, S. Haber, W.S. Stornetta, "Improving the efficiency and reliability of digital time-stamping," In *Sequences II: Methods in Communication, Security and Computer Science*, pages 329-334, 1993.

[5] S. Haber, W.S. Stornetta, "Secure names for bit-strings," In *Proceedings of the 4th ACM Conference on*

Side 7:

Computer and Communications Security, pages 28-35, April 1997.

[6] A. Back, "Hashcash - a denial of service counter-measure,"

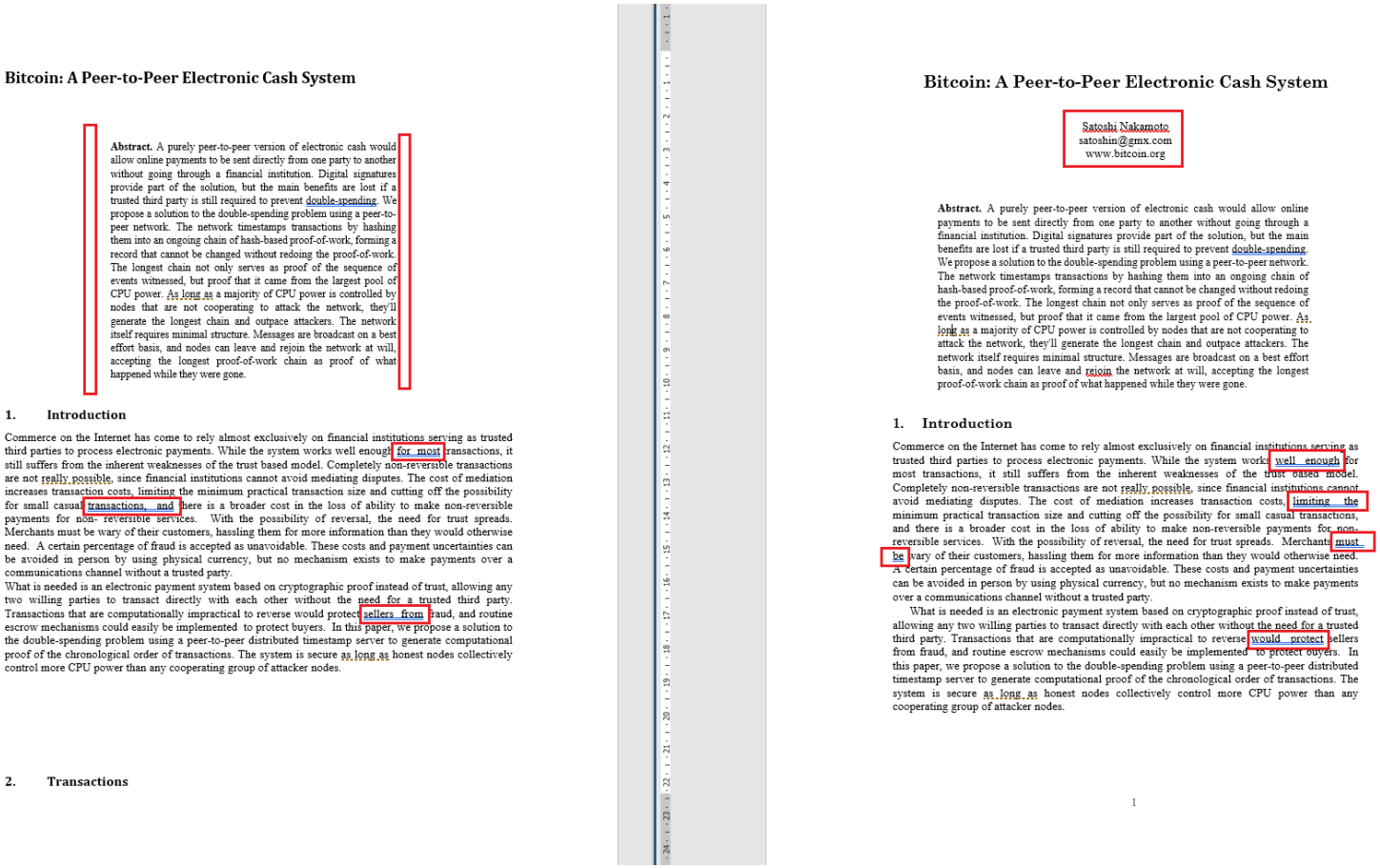
[7] R.C. Merkle, "Protocols for public key cryptosystems," In *Proc. 1980 Symposium on Security and Privacy*, IEEE Computer Society, pages 122-133, April 1980.

[8] W. Feller, "An introduction to probability theory and its applications," 1957.

4.26.3 Sammenlikning med «bitcoin-export-word.doc»

Under følger «Bilag 32.doc» (til venstre) sammenliknet med «bitcoin-export-word.docx» (til høyre).

Abstract og kapittel 1:



Bitcoin: A Peer-to-Peer Electronic Cash System

Abstract. A purely peer-to-peer version of electronic cash would allow online payments to be sent directly from one party to another without going through a financial institution. Digital signatures provide part of the solution, but the main benefits are lost if a trusted third party is still required to prevent double-spending. We propose a solution to the double-spending problem using a peer-to-peer network. The network timestamps transactions by hashing them into an ongoing chain of hash-based proof-of-work, forming a record that cannot be changed without redoing the proof-of-work. The longest chain not only serves as proof of the sequence of events witnessed, but proof that it came from the largest pool of CPU power. As long as a majority of CPU power is controlled by nodes that are not cooperating to attack the network, they'll generate the longest chain and outpace attackers. The network itself requires minimal structure. Messages are broadcast on a best effort basis, and nodes can leave and rejoin the network at will, accepting the longest proof-of-work chain as proof of what happened while they were gone.

1. Introduction

Commerce on the Internet has come to rely almost exclusively on financial institutions serving as trusted third parties to process electronic payments. While the system works well enough for most transactions, it still suffers from the inherent weaknesses of the trust based model. Completely non-reversible transactions are not really possible, since financial institutions cannot avoid mediating disputes. The cost of mediation increases transaction costs, limiting the minimum practical transaction size and cutting off the possibility for small casual transactions, and there is a broader cost in the loss of ability to make non-reversible payments for non- reversible services. With the possibility of reversal, the need for trust spreads. Merchants must be wary of their customers, hassling them for more information than they would otherwise need. A certain percentage of fraud is accepted as unavoidable. These costs and payment uncertainties can be avoided in person by using physical currency, but no mechanism exists to make payments over a communications channel without a trusted party.

What is needed is an electronic payment system based on cryptographic proof instead of trust, allowing any two willing parties to transact directly with each other without the need for a trusted third party. Transactions that are computationally impractical to reverse would protect sellers from fraud, and routine escrow mechanisms could easily be implemented to protect buyers. In this paper, we propose a solution to the double-spending problem using a peer-to-peer distributed timestamp server to generate computational proof of the chronological order of transactions. The system is secure as long as honest nodes collectively control more CPU power than any cooperating group of attacker nodes.

2. Transactions

Bitcoin: A Peer-to-Peer Electronic Cash System

Satoshi Nakamoto
satoshin@gmx.com
www.bitcoin.org

Abstract. A purely peer-to-peer version of electronic cash would allow online payments to be sent directly from one party to another without going through a financial institution. Digital signatures provide part of the solution, but the main benefits are lost if a trusted third party is still required to prevent double-spending. We propose a solution to the double-spending problem using a peer-to-peer network. The network timestamps transactions by hashing them into an ongoing chain of hash-based proof-of-work, forming a record that cannot be changed without redoing the proof-of-work. The longest chain not only serves as proof of the sequence of events witnessed, but proof that it came from the largest pool of CPU power. As long as a majority of CPU power is controlled by nodes that are not cooperating to attack the network, they'll generate the longest chain and outpace attackers. The network itself requires minimal structure. Messages are broadcast on a best effort basis, and nodes can leave and rejoin the network at will, accepting the longest proof-of-work chain as proof of what happened while they were gone.

1. Introduction

Commerce on the Internet has come to rely almost exclusively on financial institutions serving as trusted third parties to process electronic payments. While the system works well enough for most transactions, it still suffers from the inherent weaknesses of the trust based model. Completely non-reversible transactions are not really possible, since financial institutions cannot avoid mediating disputes. The cost of mediation increases transaction costs, limiting the minimum practical transaction size and cutting off the possibility for small casual transactions, and there is a broader cost in the loss of ability to make non-reversible payments for non-reversible services. With the possibility of reversal, the need for trust spreads. Merchants must be wary of their customers, hassling them for more information than they would otherwise need. A certain percentage of fraud is accepted as unavoidable. These costs and payment uncertainties can be avoided in person by using physical currency, but no mechanism exists to make payments over a communications channel without a trusted party.

What is needed is an electronic payment system based on cryptographic proof instead of trust, allowing any two willing parties to transact directly with each other without the need for a trusted third party. Transactions that are computationally impractical to reverse would protect sellers from fraud, and routine escrow mechanisms could easily be implemented to protect buyers. In this paper, we propose a solution to the double-spending problem using a peer-to-peer distributed timestamp server to generate computational proof of the chronological order of transactions. The system is secure as long as honest nodes collectively control more CPU power than any cooperating group of attacker nodes.

1366

Kapittel 2 og 3:

We define an electronic coin as a chain of digital signatures. Each owner transfers the coin to the next by digitally signing a hash of the previous transaction and the public key of the next owner and adding these to the end of the coin. A payee can verify the signatures to verify the chain of ownership.

The problem of course is the payee can't verify that one of the owners did not double-spend the coin. A common solution is to introduce a trusted central authority, or mint, that checks every transaction for double spending. After each transaction, the coin must be returned to the mint to issue a new coin, and only coins issued directly from the mint are trusted not to be double-spent. The problem with this solution is that the fate of the entire money system depends on the company running the mint, with every transaction having to go through them, just like a bank.

We need a way for the payee to know that the previous owners did not sign any earlier transactions. For our purposes, the earliest transaction is the one that counts, so we don't care about later attempts to double-spend. The only way to confirm the absence of a transaction is to be aware of all transactions. In the mint based model, the mint was aware of all transactions and decided which arrived first. To accomplish this without a trusted party, transactions must be publicly announced [1], and we need a system for participants to agree on a single history of the order in which they were received. The payee needs proof that at the time of each transaction, the majority of nodes agreed it was the first received.

3. Timestamp Server

The solution we propose begins with a timestamp server. A timestamp server works by taking a hash of a block of items to be timestamped and widely publishing the hash, such as in a newspaper or Usenet post [2-5]. The timestamp proves that the data must have existed at the time, obviously, in order to get into the hash. Each timestamp includes the previous timestamp in its hash, forming a chain, with each additional timestamp reinforcing the ones before it.

4. Proof-of-Work

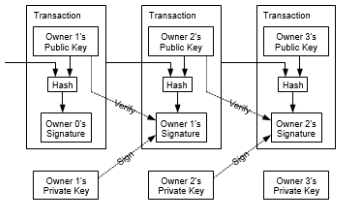
To implement a distributed timestamp server on a peer-to-peer basis, we will need to use a proof- of-work system similar to Adam Back's Hashcash [6], rather than newspaper or Usenet posts. The proof-of-work involves scanning for a value that when hashed, such as with SHA-256, the hash begins with a number of zero bits. The average work required is exponential in the number of zero bits required and can be verified by executing a single hash.

For our timestamp network, we implement the proof-of-work by incrementing a nonce in the block until a value is found that gives the block's hash the required zero bits. Once the CPU effort has been expended to make it satisfy the proof-of-work, the block cannot be changed without redoing the work. As later blocks are chained after it, the work to change the block would include redoing all the blocks after it.

The proof-of-work also solves the problem of determining representation in majority decision making. If the majority were based on one-IP-address-one-vote, it could be subverted by anyone able to allocate many IPs. Proof-of-work is essentially one-CPU-one-vote. The majority decision is represented by the longest chain, which has the greatest proof-of-work effort invested in it. If a majority of CPU power is controlled by honest nodes, the honest chain will grow the fastest and outpace any competing chains. To modify a

2. Transactions

We define an electronic coin as a chain of digital signatures. Each owner transfers the coin to the next by digitally signing a hash of the previous transaction and the public key of the next owner and adding these to the end of the coin. A payee can verify the signatures to verify the chain of ownership.

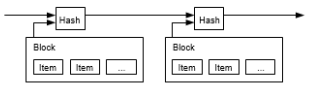


The problem of course is the payee can't verify that one of the owners did not double-spend the coin. A common solution is to introduce a trusted central authority, or mint, that checks every transaction for double spending. After each transaction, the coin must be returned to the mint to issue a new coin, and only coins issued directly from the mint are trusted not to be double-spent. The problem with this solution is that the fate of the entire money system depends on the company running the mint, with every transaction having to go through them, just like a bank.

We need a way for the payee to know that the previous owners did not sign any earlier transactions. For our purposes, the earliest transaction is the one that counts, so we don't care about later attempts to double-spend. The only way to confirm the absence of a transaction is to be aware of all transactions. In the mint based model, the mint was aware of all transactions and decided which arrived first. To accomplish this without a trusted party, transactions must be publicly announced [1], and we need a system for participants to agree on a single history of the order in which they were received. The payee needs proof that at the time of each transaction, the majority of nodes agreed it was the first received.

3. Timestamp Server

The solution we propose begins with a timestamp server. A timestamp server works by taking a hash of a block of items to be timestamped and widely publishing the hash, such as in a newspaper or Usenet post [2-5]. The timestamp proves that the data must have existed at the time, obviously, in order to get into the hash. Each timestamp includes the previous timestamp in its hash, forming a chain, with each additional timestamp reinforcing the ones before it.



1366

1367

Kapittel 4 og 5:

4. Proof-of-Work

To implement a distributed timestamp server on a peer-to-peer basis, we will need to use a proof- of-work system similar to Adam Back's Hashcash [6], rather than newspaper or Usenet posts. The proof-of-work involves scanning for a value that when hashed, such as with SHA-256, the hash begins with a number of zero bits. The average work required is exponential in the number of zero bits required and can be verified by executing a single hash.

For our timestamp network, we implement the proof-of-work by incrementing a nonce in the block until a value is found that gives the block's hash the required zero bits. Once the CPU effort has been expended to make it satisfy the proof-of-work, the block cannot be changed without redoing the work. As later blocks are chained after it, the work to change the block would include redoing all the blocks after it.

The proof-of-work also solves the problem of determining representation in majority decision making. If the majority were based on one-IP-address-one-vote, it could be subverted by anyone able to allocate many IPs. Proof-of-work is essentially one-CPU-one-vote. The majority decision is represented by the longest chain, which has the greatest proof-of-work effort invested in it. If a majority of CPU power is controlled by honest nodes, the honest chain will grow the fastest and outpace any competing chains. To modify a past block, an attacker would have to redo the proof-of-work of the block and all blocks after it and then catch up with and surpass the work of the honest nodes. We will show later that the probability of a slower attacker catching up diminishes exponentially as subsequent blocks are added.

To compensate for increasing hardware speed and varying interest in running nodes over time, the proof-of-work difficulty is determined by a moving average targeting an average number of blocks per hour. If they're generated too fast, the difficulty increases.

5. Network

The steps to run the network are as follows:

1) New transactions are broadcast to all nodes.
2) Each node collects new transactions into a block.
3) Each node works on finding a difficult proof-of-work for its block.
4) When a node finds a proof-of-work, it broadcasts the block to all nodes.
5) Nodes accept the block only if all transactions in it are valid and not already spent.
6) Nodes express their acceptance of the block by working on creating the next block in the chain, using the hash of the accepted block as the previous hash.

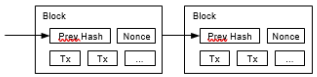
Nodes always consider the longest chain to be the correct one and will keep working on extending it. If two nodes broadcast different versions of the next block simultaneously, some nodes may receive one or the other first. In that case, they work on the first one they received, but save the other branch in case it becomes longer. The tie will be broken when the next proof- of-work is found and one branch becomes longer; the nodes that were working on the other branch will then switch to the longer one.

New transaction broadcasts do not necessarily need to reach all nodes. As long as they reach many nodes, they will get into a block before long. Block broadcasts are also tolerant of dropped messages. If a node does not receive a block, it will request it when it receives the next block and realizes it missed one.

4. Proof-of-Work

To implement a distributed timestamp server on a peer-to-peer basis, we will need to use a proof- of-work system similar to Adam Back's Hashcash [6], rather than newspaper or Usenet posts. The proof-of-work involves scanning for a value that when hashed, such as with SHA-256, the hash begins with a number of zero bits. The average work required is exponential in the number of zero bits required and can be verified by executing a single hash.

For our timestamp network, we implement the proof-of-work by incrementing a nonce in the block until a value is found that gives the block's hash the required zero bits. Once the CPU effort has been expended to make it satisfy the proof-of-work, the block cannot be changed without redoing the work. As later blocks are chained after it, the work to change the block would include redoing all the blocks after it.



The proof-of-work also solves the problem of determining representation in majority decision making. If the majority were based on one-IP-address-one-vote, it could be subverted by anyone able to allocate many IPs. Proof-of-work is essentially one-CPU-one-vote. The majority decision is represented by the longest chain, which has the greatest proof-of-work effort invested in it. If a majority of CPU power is controlled by honest nodes, the honest chain will grow the fastest and outpace any competing chains. To modify a past block, an attacker would have to redo the proof-of-work of the block and all blocks after it and then catch up with and surpass the work of the honest nodes. We will show later that the probability of a slower attacker catching up diminishes exponentially as subsequent blocks are added.

To compensate for increasing hardware speed and varying interest in running nodes over time, the proof-of-work difficulty is determined by a moving average targeting an average number of blocks per hour. If they're generated too fast, the difficulty increases.

5. Network

The steps to run the network are as follows:

1) New transactions are broadcast to all nodes.
2) Each node collects new transactions into a block.
3) Each node works on finding a difficult proof-of-work for its block.
4) When a node finds a proof-of-work, it broadcasts the block to all nodes.
5) Nodes accept the block only if all transactions in it are valid and not already spent.
6) Nodes express their acceptance of the block by working on creating the next block in the chain, using the hash of the accepted block as the previous hash.

Nodes always consider the longest chain to be the correct one and will keep working on extending it. If two nodes broadcast different versions of the next block simultaneously, some nodes may receive one or the other first. In that case, they work on the first one they received, but save the other branch in case it becomes longer. The tie will be broken when the next proof-of-work is found and one branch becomes longer; the nodes that were working on the other branch will then switch to the longer one.

3

1367

1368

Kapittel 6 og 7:

6. Incentive

By convention, the first transaction in a block is a special transaction that starts a new coin owned by the creator of the block. This adds an incentive for nodes to support the network, and provides a way to initially distribute coins into circulation, since there is no central authority to issue them. The steady addition of a constant of amount of new coins is analogous to gold miners expending resources to add gold to circulation. In our case, it is CPU time and electricity that is expended.

The incentive can also be funded with transaction fees. If the output value of a transaction is less than its input value, the difference is a transaction fee that is added to the incentive value of the block containing the transaction. Once a predetermined number of coins have entered circulation, the incentive can transition entirely to transaction fees and be completely inflation free.

The incentive may help encourage nodes to stay honest. If a greedy attacker is able to assemble more CPU power than all the honest nodes, he would have to choose between using it to defraud people by stealing back his payments, or using it to generate new coins. He ought to find it more profitable to play by the rules, such rules that favour him with more new coins than everyone else combined, than to undermine the system and the validity of his own wealth.

7. Reclaiming Disk Space

Once the latest transaction in a coin is buried under enough blocks, the spent transactions before it can be discarded to save disk space. To facilitate this without breaking the block's hash, transactions are hashed in a Binary Tree [7][2][5], with only the root included in the block's hash. Old blocks can then be compacted by stubbing off branches of the tree. The interior hashes do not need to be stored.

A block header with no transactions would be about 80 bytes. If we suppose blocks are generated every 10 minutes, 80 bytes * 6 * 24 * 365 = 4.2MB per year. With computer systems typically selling with 2GB of RAM as of 2008, and Moore's Law predicting current growth of 1.2GB per year, storage should not be a problem even if the block headers must be kept in memory.

8. Simplified Payment Verification

It is possible to verify payments without running a full network node. A user only needs to keep a copy of the block headers of the longest proof-of-work chain, which he can get by querying network nodes until he's convinced he has the longest chain, and obtain the branch of the binary tree [7] linking the transaction to the block it's timestamped in. He can't check the transaction for himself, but by linking it to a place in the chain, he can see that a network node has accepted it, and blocks added after it further confirm the network has accepted it.

As such, the verification is reliable as long as honest nodes control the network, but is more vulnerable if the network is overpowered by an attacker. While network nodes can verify transactions for themselves, the simplified method can be fooled by an attacker's fabricated transactions for as long as the attacker can continue to overpower the network. One strategy to protect against this would be to accept alerts from network nodes when they detect an invalid block, prompting the user's software to download the full block and alerted transactions to confirm the inconsistency. Businesses that receive frequent payments will probably still want to run their own nodes for more independent security and quicker verification.

6. Incentive

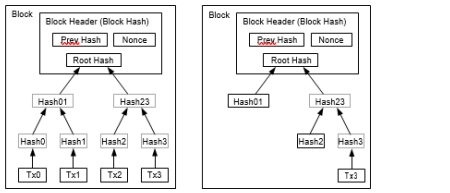
By convention, the first transaction in a block is a special transaction that starts a new coin owned by the creator of the block. This adds an incentive for nodes to support the network, and provides a way to initially distribute coins into circulation, since there is no central authority to issue them. The steady addition of a constant of amount of new coins is analogous to gold miners expending resources to add gold to circulation. In our case, it is CPU time and electricity that is expended.

The incentive can also be funded with transaction fees. If the output value of a transaction is less than its input value, the difference is a transaction fee that is added to the incentive value of the block containing the transaction. Once a predetermined number of coins have entered circulation, the incentive can transition entirely to transaction fees and be completely inflation free.

The incentive may help encourage nodes to stay honest. If a greedy attacker is able to assemble more CPU power than all the honest nodes, he would have to choose between using it to defraud people by stealing back his payments, or using it to generate new coins. He ought to find it more profitable to play by the rules, such rules that favour him with more new coins than everyone else combined, than to undermine the system and the validity of his own wealth.

7. Reclaiming Disk Space

Once the latest transaction in a coin is buried under enough blocks, the spent transactions before it can be discarded to save disk space. To facilitate this without breaking the block's hash, transactions are hashed in a Merkle Tree [7][2][5], with only the root included in the block's hash. Old blocks can then be compacted by stubbing off branches of the tree. The interior hashes do not need to be stored.



Transactions Hashed in a Merkle Tree | After Pruning Tx0-2 from the Block

A block header with no transactions would be about 80 bytes. If we suppose blocks are generated every 10 minutes, 80 bytes * 6 * 24 * 365 = 4.2MB per year. With computer systems typically selling with 2GB of RAM as of 2008, and Moore's Law predicting current growth of 1.2GB per year, storage should not be a problem even if the block headers must be kept in memory.

4

1368

1369

Kapittel 8 og 9:

8. Simplified Payment Verification

It is possible to verify payments without running a full network node. A user only needs to keep a copy of the block headers of the longest proof-of-work chain, which he can get by querying network nodes until he's convinced he has the longest chain, and obtain the branch of the binary tree [7] linking the transaction to the block it's timestamped in. He can't check the transaction for himself, but by linking it to a place in the chain, he can see that a network node has accepted it, and blocks added after it further confirm the network has accepted it.

As such, the verification is reliable as long as honest nodes control the network, but is more vulnerable if the network is overpowered by an attacker. While network nodes can verify transactions for themselves, the simplified method can be fooled by an attacker's fabricated transactions for as long as the attacker can continue to overpower the network. One strategy to protect against this would be to accept alerts from network nodes when they detect an invalid block, prompting the user's software to download the full block and alerted transactions to confirm the inconsistency. Businesses that receive frequent payments will probably still want to run their own nodes for more independent security and quicker verification.

9. Combining and Splitting Value

Although it would be possible to handle coins individually, it would be unwieldy to make a separate transaction for every cent in a transfer. To allow value to be split and combined, transactions contain multiple inputs and outputs. Normally there will be either a single input from a larger previous transaction or multiple inputs combining smaller amounts, and at most two outputs: one for the payment, and one returning the change, if any, back to the sender.

It should be noted that fan-out, where a transaction depends on several transactions, and those transactions depend on many more, is not a problem here. There is never the need to extract a complete standalone copy of a transaction's history.

10. Privacy

The traditional banking model achieves a level of privacy by limiting access to information to the parties involved and the trusted third party. The necessity to announce all transactions publicly precludes this method, but privacy can still be maintained by breaking the flow of information in another place: by keeping public keys anonymous. The public can see that someone is sending an amount to someone else, but without information linking the transaction to anyone. This is similar to the level of information released by stock exchanges, where the time and size of individual trades, the "tape", is made public, but without telling who the parties were.

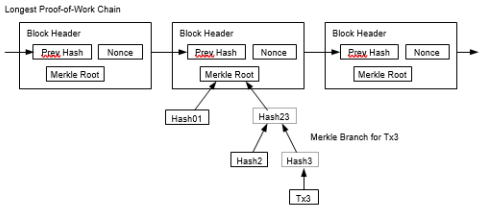
As an additional firewall, a new key pair should be used for each transaction to keep them from being linked to a common owner. Some linking is still unavoidable with multi-input transactions, which necessarily reveal that their inputs were owned by the same owner. The risk is that if the owner of a key is revealed, linking could reveal other transactions that belonged to the same owner.

11. Calculations

We consider the scenario of an attacker trying to generate an alternate chain faster than the honest chain. Even if this is accomplished, it does not throw the system open to arbitrary changes, such as creating value out of thin air or taking money that never belonged to the attacker. Nodes are not going to accept an invalid transaction as payment, and honest nodes will never accept a block containing them. An attacker can only try to change one of his own transactions to take back money he recently spent.
The race between the honest chain and an attacker chain can be characterized as a Binomial Random Walk.

8. Simplified Payment Verification

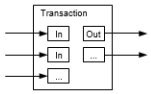
It is possible to verify payments without running a full network node. A user only needs to keep a copy of the block headers of the longest proof-of-work chain, which he can get by querying network nodes until he's convinced he has the longest chain, and obtain the Merkle branch linking the transaction to the block it's timestamped in. He can't check the transaction for himself, but by linking it to a place in the chain, he can see that a network node has accepted it, and blocks added after it further confirm the network has accepted it.



As such, the verification is reliable as long as honest nodes control the network, but is more vulnerable if the network is overpowered by an attacker. While network nodes can verify transactions for themselves, the simplified method can be fooled by an attacker's fabricated transactions for as long as the attacker can continue to overpower the network. One strategy to protect against this would be to accept alerts from network nodes when they detect an invalid block, prompting the user's software to download the full block and alerted transactions to confirm the inconsistency. Businesses that receive frequent payments will probably still want to run their own nodes for more independent security and quicker verification.

9. Combining and Splitting Value

Although it would be possible to handle coins individually, it would be unwieldy to make a separate transaction for every cent in a transfer. To allow value to be split and combined, transactions contain multiple inputs and outputs. Normally there will be either a single input from a larger previous transaction or multiple inputs combining smaller amounts, and at most two outputs: one for the payment, and one returning the change, if any, back to the sender.



It should be noted that fan-out, where a transaction depends on several transactions, and those transactions depend on many more, is not a problem here. There is never the need to extract a complete standalone copy of a transaction's history.

5

1369

1370

Kapittel 10 og 11:

10. Privacy

The traditional banking model achieves a level of privacy by limiting access to information to the parties involved and the trusted third party. The necessity to announce all transactions publicly precludes this method, but privacy can still be maintained by breaking the flow of information in another place: by keeping public keys anonymous. The public can see that someone is sending an amount to someone else, but without information linking the transaction to anyone. This is similar to the level of information released by stock exchanges, where the time and size of individual trades, the "tape", is made public, but without telling who the parties were.

As an additional firewall, a new key pair should be used for each transaction to keep them from being linked to a common owner. Some linking is still unavoidable with multi-input transactions, which necessarily reveal that their inputs were owned by the same owner. The risk is that if the owner of a key is revealed, linking could reveal other transactions that belonged to the same owner.

11. Calculations

We consider the scenario of an attacker trying to generate an alternate chain faster than the honest chain. Even if this is accomplished, it does not throw the system open to arbitrary changes, such as creating value out of thin air or taking money that never belonged to the attacker. Nodes are not going to accept an invalid transaction as payment, and honest nodes will never accept a block containing them. An attacker can only try to change one of his own transactions to take back money he recently spent.

The race between the honest chain and an attacker chain can be characterized as a Binomial Random Walk. The success event is the honest chain being extended by one block, increasing its lead by +1, and the failure event is the attacker's chain being extended by one block, reducing the gap by -1.

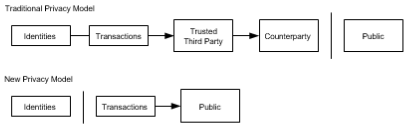
The probability of an attacker catching up from a given deficit is analogous to a Gambler's Ruin problem. Suppose a gambler with unlimited credit starts at a deficit and plays potentially an infinite number of trials to try to reach breakeven. We can calculate the probability he ever reaches breakeven, or that an attacker ever catches up with the honest chain, as follows [8]:

p = probability an honest node finds the next block
q = probability the attacker finds the next block
q_z = probability the attacker will ever catch up from z blocks behind

$$q_z = \begin{cases} 1 & \text{if } p \leq q \\ (q/p)^z & \text{if } p > q \end{cases}$$

10. Privacy

The traditional banking model achieves a level of privacy by limiting access to information to the parties involved and the trusted third party. The necessity to announce all transactions publicly precludes this method, but privacy can still be maintained by breaking the flow of information in another place: by keeping public keys anonymous. The public can see that someone is sending an amount to someone else, but without information linking the transaction to anyone. This is similar to the level of information released by stock exchanges, where the time and size of individual trades, the "tape", is made public, but without telling who the parties were.



As an additional firewall, a new key pair should be used for each transaction to keep them from being linked to a common owner. Some linking is still unavoidable with multi-input transactions, which necessarily reveal that their inputs were owned by the same owner. The risk is that if the owner of a key is revealed, linking could reveal other transactions that belonged to the same owner.

11. Calculations

We consider the scenario of an attacker trying to generate an alternate chain faster than the honest chain. Even if this is accomplished, it does not throw the system open to arbitrary changes, such as creating value out of thin air or taking money that never belonged to the attacker. Nodes are not going to accept an invalid transaction as payment, and honest nodes will never accept a block containing them. An attacker can only try to change one of his own transactions to take back money he recently spent.

The race between the honest chain and an attacker chain can be characterized as a Binomial Random Walk. The success event is the honest chain being extended by one block, increasing its lead by +1, and the failure event is the attacker's chain being extended by one block, reducing the gap by -1.

The probability of an attacker catching up from a given deficit is analogous to a Gambler's Ruin problem. Suppose a gambler with unlimited credit starts at a deficit and plays potentially an infinite number of trials to try to reach breakeven. We can calculate the probability he ever reaches breakeven, or that an attacker ever catches up with the honest chain, as follows [8]:

p = probability an honest node finds the next block
q = probability the attacker finds the next block
q_z = probability the attacker will ever catch up from z blocks behind

$$q_z = \begin{cases} 1 & \text{if } p \leq q \\ (q/p)^z & \text{if } p > q \end{cases}$$

1370

1371

Kapittel 11 forts.:

Given our assumption that $p > q$, the probability drops exponentially as the number of blocks the attacker has to catch up with increases. With the odds against him, if he doesn't make a lucky lunge forward early on, his chances become vanishingly small as he falls further behind.

We now consider how long the recipient of a new transaction needs to wait before being sufficiently certain the sender can't change the transaction. We assume the sender is an attacker who wants to make the recipient believe he paid him for a while, then switch it to pay back to himself after some time has passed. The receiver will be alerted when that happens, but the sender hopes it will be too late.

The receiver generates a new key pair and gives the public key to the sender shortly before signing. This prevents the sender from preparing a chain of blocks ahead of time by working on it continuously until he is lucky enough to get far enough ahead, then executing the transaction at that moment. Once the transaction is sent, the dishonest sender starts working in secret on a parallel chain containing an alternate version of his transaction.

The recipient waits until the transaction has been added to a block and z blocks have been linked after it. He doesn't know the exact amount of progress the attacker has made, but assuming the honest blocks took the average expected time per block, the attacker's potential progress will be a Poisson distribution with expected value:

$$\lambda = z\frac{q}{p}$$

To get the probability the attacker could still catch up now, we multiply the Poisson density for each amount of progress he could have made by the probability he could catch up from that point:

$$\sum_{k=0}^{\infty} \frac{\lambda^k e^{-\lambda}}{k!} \cdot \begin{cases} (q/p)^{(z-k)} & \text{if } k \le z \\ 1 & \text{if } k > z \end{cases}$$

Rearranging to avoid summing the infinite tail of the distribution...

$$1 - \sum_{k=0}^{z} \frac{\lambda^k e^{-\lambda}}{k!} \left(1 - (q/p)^{(z-k)}\right)$$

Converting to C code...

```
#include <math.h>
double AttackerSuccessProbability(double q, int z)
{
    double p = 1.0 - q;
    double lambda = z * (q / p);
    double sum = 1.0;
    int i, k;
    for (k = 0; k <= z; k++)
    {
        double poisson = exp(-lambda);
        for (i = 1; i <= k; i++)
            poisson *= lambda / i;
        sum -= poisson * (1 - pow(q / p, z - k));
    }
    return sum;
}
```


1371

1372

Kapittel 12:

Running some results, we can see the probability drop off exponentially with z.

```
q=0.1
z=0     P=1.000000
z=1     P=0.204587
z=2     P=0.050978
z=3     P=0.013172
z=4     P=0.003455
z=5     P=0.000914
z=6     P=0.000243
z=7     P=0.000065
z=8     P=0.000017
z=9                         P=0.0000046 z=10 P=0.0000012

q=0.3
z=0                         P=1.0000000
z=5  P=0.1773523 z=10  P=0.0416605 z=15  P=0.0101008 z=20  P=0.0024804 z=25  P=0.0006132 z=30
P=0.0001522 z=35  P=0.0000379 z=40  P=0.0000095 z=45  P=0.0000024 z=50  P=0.0000006
```

Solving for P less than 0.1%...

```
P < 0.001 q=0.10              z=5
q=0.15    z=8
q=0.20    z=11
q=0.25    z=15
q=0.30    z=24
q=0.35    z=41
q=0.40    z=89
q=0.45    z=340
```

12. Conclusion

We have proposed a system for electronic transactions without relying on trust. We started with the usual framework of coins made from digital signatures, which provides strong control of ownership, but is incomplete without a way to prevent double-spending. To solve this, we proposed a peer-to-peer network using proof-of-work to record a public history of transactions that quickly becomes computationally impractical for an attacker to change if honest nodes control a majority of CPU power. The network is robust in its unstructured simplicity. Nodes work all at once with little coordination. They do not need to be identified, since messages are not routed to any particular place and only need to be delivered on a best effort basis. Nodes can leave and rejoin the network at will, accepting the proof-of-work chain as proof of what happened while they were gone. They vote with their CPU power, expressing their acceptance of valid blocks by working on extending them and rejecting invalid blocks by refusing to work on them. Any needed rules and incentives can be enforced with this consensus mechanism.

References

[1] W.Dai, "b-money," http://www.weidai.com/bmoney.txt, 1998.

[2] H. Massias, X.S. Avila, and J.-J. Quisquater, "Design of a secure timestamping service with minimal trust requirements," In *20th Symposium on Information Theory in the Benelux*, May 1999.

[3] S. Haber, W.S. Stornetta, "How to time-stamp a digital document," In *Journal of Cryptology*, vol 3,no 2, pages 99-111, 1991.

[4] D. Bayer, S. Haber, W.S. Stornetta, "Improving the efficiency and reliability of digitaltime-stamping," In *Sequences II: Methods in Communication, Security and Computer Science*, pages 329-334, 1993.

[5] S. Haber, W.S. Stornetta, "Secure names for bit-strings," In *Proceedings of the 4th ACM Conference on*

Running some results, we can see the probability drop off exponentially with z.

```
q=0.1
z=0     P=1.0000000
z=1     P=0.2045873
z=2     P=0.0509779
z=3     P=0.0131722
z=4     P=0.0034552
z=5     P=0.0009137
z=6     P=0.0002428
z=7     P=0.0000647
z=8     P=0.0000173
z=9     P=0.0000046
z=10    P=0.0000012

q=0.3
z=0     P=1.0000000
z=5     P=0.1773523
z=10    P=0.0416605
z=15    P=0.0101008
z=20    P=0.0024804
z=25    P=0.0006132
z=30    P=0.0001522
z=35    P=0.0000379
z=40    P=0.0000095
z=45    P=0.0000024
z=50    P=0.0000006
```

Solving for P less than 0.1%...

```
P < 0.001
q=0.10    z=5
q=0.15    z=8
q=0.20    z=11
q=0.25    z=15
q=0.30    z=24
q=0.35    z=41
q=0.40    z=89
q=0.45    z=340
```

12. Conclusion

We have proposed a system for electronic transactions without relying on trust. We started with the usual framework of coins made from digital signatures, which provides strong control of ownership, but is incomplete without a way to prevent double-spending. To solve this, we proposed a peer-to-peer network using proof-of-work to record a public history of transactions that quickly becomes computationally impractical for an attacker to change if honest nodes control a majority of CPU power. The network is robust in its unstructured simplicity. Nodes work all at once with little coordination. They do not need to be identified, since messages are not routed to any particular place and only need to be delivered on a best effort basis. Nodes can leave and rejoin the network at will, accepting the proof-of-work chain as proof of what happened while they were gone. They vote with their CPU power, expressing their acceptance of valid blocks by working on extending them and rejecting invalid blocks by refusing to work on them. Any needed rules and incentives can be enforced with this consensus mechanism.

1372

1373

References:

References

[1] W. Dai, "b-money," http://www.weidai.com/bmoney.txt, 1998.

[2] H. Massias, X.S. Avila, and J.-J. Quisquater, "Design of a secure timestamping service with minimal trust requirements," In *20th Symposium on Information Theory in the Benelux*, May 1999.

[3] S. Haber, W.S. Stornetta, "How to time-stamp a digital document," In *Journal of Cryptology*, vol 3, no 2, pages 99-111, 1991.

[4] D. Bayer, S. Haber, W.S. Stornetta, "Improving the efficiency and reliability of digital time-stamping," In *Sequences II: Methods in Communication, Security and Computer Science*, pages 329-334, 1993.

[5] S. Haber, W.S. Stornetta, "Secure names for bit-strings," In *Proceedings of the 4th ACM Conference on Computer and Communications Security*, pages 28-35, April 1997.

[6] A. Back, "Hashcash - a denial of service counter-measure,"

[7] R.C. Merkle, "Protocols for public key cryptosystems," In *Proc. 1980 Symposium on Security and Privacy*, IEEE Computer Society, pages 122-133, April 1980.

[8] W. Feller, "An introduction to probability theory and its applications," 1957.

References

[1] W. Dai, "b-money," http://www.weidai.com/bmoney.txt, 1998.

[2] H. Massias, X.S. Avila, and J.-J. Quisquater, "Design of a secure timestamping service with minimal trust requirements," In *20th Symposium on Information Theory in the Benelux*, May 1999.

[3] S. Haber, W.S. Stornetta, "How to time-stamp a digital document," In *Journal of Cryptology*, vol 3, no 2, pages 99-111, 1991.

[4] D. Bayer, S. Haber, W.S. Stornetta, "Improving the efficiency and reliability of digital time-stamping," In *Sequences II: Methods in Communication, Security and Computer Science*, pages 329-334, 1993.

[5] S. Haber, W.S. Stornetta, "Secure names for bit-strings," In *Proceedings of the 4th ACM Conference on Computer and Communications Security*, pages 28-35, April 1997.

[6] A. Back, "Hashcash - a denial of service counter-measure," http://www.hashcash.org/papers/hashcash.pdf, 2002.

[7] R.C. Merkle, "Protocols for public key cryptosystems," In *Proc. 1980 Symposium on Security and Privacy*, IEEE Computer Society, pages 122-133, April 1980.

[8] W. Feller, "An introduction to probability theory and its applications," 1957.

1373

4.27 Vedlegg 27: Font-oversikt fra tekst: Referansefiler

Font-oversikt: Referansefiler			
Fontfamilie	**bitcoin.pdf**	**bitcoin-draft.pdf**	**SSRN-id3440802.pdf**
Arial	**ArialMT** Type: TrueType Encoding: Built-in	**ArialMT** Type: TrueType Encoding: Built-in	**ArialMT** Type: TrueType Encoding: Built-in
			ArialMT Type: TrueType (CID) Encoding: Identity-H
Calibri			
Cambria			
CenturySchoolbook	**CenturySchoolbook-Bold** Type: TrueType Encoding: Built-in	**CenturySchoolbook-Bold** Type: TrueType Encoding: Built-in	**CenturySchoolbook-Bold** Type: TrueType Encoding: Built-in
			CenturySchoolbook-Bold Type: TrueType (CID) Encoding: Identity-H
Courier New	**CourierNewPSMT** Type: TrueType Encoding: Built-in	**CourierNewPSMT** Type: TrueType Encoding: Built-in	**CourierNewPSMT** Type: TrueType Encoding: Built-in
Helvetica			**Helvetica** Type: Type 1 Encoding: Ansi Actual Font: ArialMT Actual Font Type: TrueType
Open Symbol	**OpenSymbol** Type: TrueType Encoding: Built-in	**OpenSymbol** Type: TrueType Encoding: Built-in	**OpenSymbol** Type: TrueType Encoding: Built-in
Lucida Sans			
Times New Roman	**TimesNewRomanPS-BoldMT** Type: TrueType Encoding: Built-in	**TimesNewRomanPS-BoldMT** Type: TrueType Encoding: Built-in	**TimesNewRomanPS-BoldMT** Type: TrueType Encoding: Built-in
	TimesNewRomanPS-ItalicMT Type: TrueType Encoding: Built-in	**TimesNewRomanPS-ItalicMT** Type: TrueType Encoding: Built-in	**TimesNewRomanPS-ItalicMT** Type: TrueType Encoding: Built-in
	TimesNewRomanPSMT Type: TrueType Encoding: Built-in	**TimesNewRomanPSMT** Type: TrueType Encoding: Built-in	**TimesNewRomanPSMT** Type: TrueType Encoding: Built-in
			TimesNewRomanPSMT Type: TrueType (CID) Encoding: Identity-H

4.28 Vedlegg 28: Font-oversikt fra tekst: Bilag 27, 28 & 32

Font-oversikt: Bilag 27, 28 & 32				
Font-familie	**Bilag27.odt**	**2008-05-06_B027_-_TimeCoin_Peer-to-Peer_Electronic_Cash_System__Dr._Craig_S._Wright.pdf.pdf**	**Bilag28.pdf**	**Bilag32.doc**
Arial	**Arial**	**Arial** Type: TrueType Encoding: Ansi	**ArialMT** Type: TrueType Encoding: Built-in	**Arial** Type: TrueType Encoding: Ansi
		ArialMT Type: TrueType (CID) Encoding: Identity-H	**ArialMT** Type: TrueType (CID) Encoding: Identity-H	**ArialMT** Type: TrueType Encoding: Ansi
Calibri		**Calibri** Type: True Type (CID) Encoding: Identity-H		
Cambria	**Cambria Math**			**Cambria** Type: TrueType Encoding: Ansi
				Cambria-Bold Type: TrueType Encoding: Ansi
				Cambria-Italic Type: TrueType Encoding: Ansi
CenturySchoolbook	**CenturySchoolbook**	**CenturySchoolbook** Type: TrueType Encoding: Ansi	**CenturySchoolbook-Bold** Type: TrueType Encoding: Built-in	
	CenturySchoolbook-Bold	**CenturySchoolbook-Bold** Type: TrueType (CID) Encoding: Ansi	**CenturySchoolbook-Bold** Type: TrueType (CID) Encoding: Identity-H	
	CenturySchoolbook-Italic	**CenturySchoolbook-Italic** Type: TrueType (CID) Encoding: Ansi		
Courier New	**CourierNew**	**CourierNewPSMT** Type: TrueType Encoding: Ansi	**CourierNewPSMT** Type: TrueType Encoding: Built-in	**CourierNewPSMT** Type: TrueType Encoding: Ansi
Hel-vetica				
Open Symbol			**OpenSymbol** Type: TrueType Encoding: Built-in	
Lucida Sans				**LucidaSansUnicode** Type: TrueType Encoding: Ansi
Times New Roman	**Times New Roman - Bold**	**TimesNewRomanPS-BoldMT** Type: TrueType (CID) Encoding: Identity-H	**TimesNewRomanPS-BoldMT** Type: TrueType Encoding: Built-in	**TimesNewRomanPS-BoldMT** Type: TrueType Encoding: Ansi
	Times New Roman	**TimesNewRomanPSMT** Type: TrueType (CID) Encoding: Identity-H	**TimesNewRomanPS-ItalicMT** Type: TrueType Encoding: Built-in	**TimesNewRomanPS-ItalicMT** Type: TrueType Encoding: Ansi
			TimesNewRomanPSMT Type: TrueType Encoding: Built-in	**TimesNewRomanPSMT** Type: TrueType Encoding: Ansi
		TimesNewRomanPSMT Type: TrueType (CID) Encoding: Identity-H		

4.29 Vedlegg 29: Bilag med kildekode

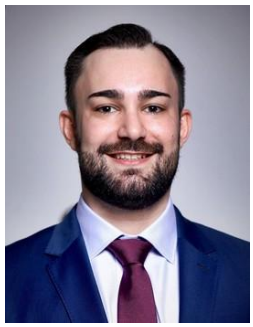
Kildekode: Bilagsoversikt		
Bilagsnr WR	**Opprinnelig filnavn**	**MD5-hash**
Bilag 35, 52, 58	base58.h	4CBF8991ED978E0444A17BDE6CD6C675
Bilag 36, 53, 59	bignum.h	22823C178AF18695224EC162BABF057A
Bilag 37, 60	db.cpp	971D002AA007AB2C577FA3DB5D234767
Bilag 38, 61	db.h	D702E77C555B184E4AA5C49B54FF1BD2
Bilag 39, 63	irc.cpp	E41E8555D0B853821EC4D3AB0668E6EF
Bilag 40, 72	net.cpp	040A5FD6BD38B862B85B39D5743E00F4
Bilag 41, 89	util.h	D6E55DA47374A67B0CF1EC69E344ADE6
Bilag 42, 88	util.cpp	25A514B43A3AB03A940EE4F6153D648A
Bilag 43, 86	uint256.h	D436DEC4D6ECB68E812B7BEC53591C01
Bilag 44, 65	key.h	1E9AEE11EDE3B43563BCB642FABA4333
Bilag 45, 70	market.cpp	6C9BAFB7EE6867CA9FC32F5F63A684F1
Bilag 46, 71	market.h	3A2001CFB55CD4ECC71CC839DF95F929
Bilag 47, 75	readme.txt	30EC76D492A532695AD074540E3CB114
Bilag 48, 87	uiproject.fbp	EF8A605E21C5A1C8C97B1717E1929DE7
Bilag 51	readme.txt	55C076538B7DD357549FEEFFF1E2BEF1
Bilag 62	headers.h	21336A2D1E0E78A99FEEEA04B97F8493
Bilag 64	irc.h	47C918062348E4B2F296282624C89650
Bilag 66	license.txt	96F9785634054DF9927715FEB0AC3D3B
Bilag 67	main.cpp	E04B403D08684267AD12CA71D244BFD4
Bilag 68	makefile	6915D8E33CD93261EB9D1272E3799EC0
Bilag 69	makefile.vc	DF318DCEF323D9AD356C456FCF40B609
Bilag 73	net.h	2F0786BBC70A7F46C1144BB3D0ABF6CA
Bilag 76	script.cpp	768017113C0F7D8AFB6D3A7CA93E1F66
Bilag 77	script.h	F4C66945D0B7C0116F6254D4E6F747DF
Bilag 78	serialize.h	B3032E31BCABECB9EDE4198BAE4BCF95
Bilag 79	sha.cpp	4E7AF8CE43B6AFE3D3C12BDE38DCA123
Bilag 80	sha.h	D18A1BC8F860C245A835D4920297FAB0
Bilag 81	ui.cpp	EBD8CB8ADB86AE3D8CA5EB86A69DB992
Bilag 82	ui.h	DEC738C0C476E280073B6425BE8AAC89
Bilag 83	ui.rc	F0119F58B4AB8D298AFB1D9A67CDA050
Bilag 84	uibase.cpp	7A0B4E31F2AE6FEB08BB38116BCD8549
Bilag 85	uibase.h	25F171C6C3646E91BA8AD4914AE65AE2

4.30 Vedlegg 30: Kort CV

Navn:	Lars Wilberg	
Tittel:	Partner, KPMG Forensic Technology/Cyber & Security	
Bakgrunn:	Lars Wilberg er partner i KPMG. Lars har mer enn 25 års erfaring fra digital etterforskning, hvorav 13 av de fra politiet, spesielt i Økokrim og Nasjonalt datakrimsenter, og 15 av de fra granskinger i det private næringsliv. Lars har tidligere vært oppnevnt som sakkyndig i en rekke større rettstvister, senest i Høyesterett i en sak om datasikring.	

Navn:	Bjørn Krok	
Tittel:	Direktør, KPMG Forensic Technology	
Bakgrunn:	Bjørn Krok er direktør i KPMG Forensic Technology og har mer enn 25 års erfaring innen IT-sikkerhet, computer forensics, eDiscovery og granskinger. Hans ekspertise ligger spesielt innen sikring og analyse av digital informasjon og avansert computer forensics. Han har vært oppnevnt som sakkyndig i et antall rettstvister i ulike rettsinstanser.	

Navn:	Matija Puzar	
Tittel:	Senior Manager, KPMG Cyber & Security	
Bakgrunn:	Matija Puzar er Senior Manager i KPMGs Cyber & Security. Han har siden 2016 hatt hovedfokus på IT-sikkerhet på operativt nivå, fra penetrasjonstesting og design av systemer og protokoller, til kursholdning og rådgivning. Han har i tillegg over 20 års erfaring med utvikling på diverse plattformer. Matija har en PHD fra Universitet i Oslo.	

Navn:	Karl Stefan Afradi	
Tittel:	Manager, KPMG Cyber & Security	
Bakgrunn:	Karl Stefan Afradi har jobbet som fagekspert i KPMG i flere prosjekter innenfor cybersikkerhet siden 2017. Karl Stefan er sertifisert og akkreditert innenfor europeisk personvernlovgivning (CIPP/E) og informasjonssikkerhet (ISO 27001).	

1379

Navn:	Emil Bjørnstad	
Tittel:	Manager, KPMG Forensic Technology	
Bakgrunn:	Emil Bjørnstad har jobbet i KPMG siden 2016 og innehar bred kompetanse innenfor eDiscovery og tekst- og innholdsanalyser. Emil innehar sin ekspertise i særdeleshet innen sikring og analyse av elektroniske spor. Emil har en dobbel mastergrad fra Norges Handelshøyskole og Louvain School of Management.	