UTXO based Certificates (UTXOC) ‘you-chi-ock’

MiWCryptoCurrency – miwcryptocurrency@gmail.com

Abstract

Proof of concept cryptocurrency (bitcoin-like) private keys used to generate valid signed ECDSA with-SHA256 x509 certificates to be used for medium term authentication within the TLS protocol and other and future applications. These certificates have additional explicit value by merit that they prove ownership of decentralised bitcoin-like value. This construction has been formatively titled **U**nspent Transa**X**tion **O**utput **C**ertificate or UTXOC, pronounced ‘you-chi-ock’

Principal

What does this provide in addition to every-day Trusted CA Certified certificates we know and love from such trusted names as Honest Achmed (HA2011) and Diginotar (BlackTulip 2011)?

This paper argues that by appending a *stored value* associated with a prior bitcoin-like transaction to a private key, and formalising this binding in an x509 container, the resultant certificate is more valuable than a standard x509 certificate.

This relationship can be used to formalise Enhanced Key Usage or Certificate Policies in x509, high valued keys (e.g.: Internet Banking https keys) might require a large *stored value* to be valid.

The cost of a Man-in-the-middle TLS certificate substitution attack on UTXOC is higher than standard certificates. If a client is expecting a sufficiently valued UTXOC, and is presented with a lesser valued or standard certificate this would be detectable.

It also provides independently verifiable proof of key compromise in that an attacker is likely to spend any stored value associated with a private key if they compromise a TLS Server Authentication keys, SSH keys, Client Certificate keys, etc. An opportunistic thief will take what is left in plain sight.

In addition, the certificate life validity can act as a type of *cryptocurrency bearer bond* in. (e.g.: For the bond to be valid, the unspent output in the transaction referenced in the certificate must remain unspent).After the *valid to* date has expired, this transaction can be claimed and the value transferred to another address (i.e: private key) to be used for a new certificate or spent.

Background – Bitcoin like cryptocurrencies

Bitcoin is a purely peer-to-peer electronic cash solution created by Satoshi Nakamoto (BTC 2008) and released in early 2009. At the time of writing, the total market capitalization of Bitcoin is about 8 billion United States Dollars. The first ‘alt-coin’, Namecoin, was created using the bitcoin codebase to implement a decentralised DNS-like system and light directory.

Experimentation with alternative hash functions was first tested with the Tenebrix CryptoCurrency; Although this particular currency had limited adoption as of 2014, it paved the way for the second highest market cap valued cryptocurrency, Litecoin.

Litecoin was created Charlie Lee in 2011. Like Tenebrix, it offered different time, reward, and proof-of-work parameters to the bitcoin model; its project goal was said to be Silver to Bitcoins Gold.

Litecoin differed from Bitcoin in that it used a different proof-of-work hash function, known as SCRYPT (SCRYPT, 2012). This hash function, more specifically a Key Derivation Function, (Draft-Scrypt-RFC-2012), was invented by Colin Percival to help secure his TarSnap online backup service. SCRYPT was chosen over the double SHA256 hash used by Bitcoin because at the time it was belived that the memory hard nature of this algorithm would limit the ability for low cost, commodity mining hardware, especially with the of the emergence of specialty bitcoin mining (hashing) ASICS.

The SCRYPT algorithm lead to what was initially a called a ‘CPU-only’, or at least ‘ASIC-resistant’ coins. Over time, more efficient SCRYPT hashing kernels were created to be run on Graphic Processing Units. For the next few years, PC mining rigs, or a headless computer system with a number of high powered graphics cards, were a popular way to mine Litecoin and other derivatives.

This in turn encouraged the creation of countless SCRYPT-based Litecoin derivatives with alternative parameters for block time, reward. Commodity SCRYPT hashing hardware is available from a number of vendors today. In later 2013, the currency Dogecoin was launched; initially as a parody of the multitude of new cryptocurrency derivatives. By mid 2014, this coin had a market capitalization of over 50 million USD.

The scrypt-adaptive-n hash function, was created for the Vertcoin cryptocurrency in 2014. This algorithm differs to from standard scrypt by having the network increment the n value in scrypt, to increase memory requirements of mining over time.

Its authors belive that eventually the memory requirements of a large n value will make GPU mining cost prohibitive; returning to CPU based mining on the relatively cheaper system memory.

While the SCRYPT based cryptocurerncies were maturing, composite hash function cryptocurrencies were proposed, employing a chain of alternative hash functions in sequence to both increase the computational complexity of the proof-of-work function and *hedge the bet* against ASIC development targeting a particular hash function. Popular families of composite cryptocurrencies include Quark, X11 (Xcurrency, Blackcoin) and X13 (Marucoin); employing 9, 11 and 13 composite hash functions respectively.

Bitcoin address generation

An bitcoin address is formed by generating an EC private key. An EC private key is defined as an unsigned integer of byte length *l* in the range 0 to log2(*n*), where *n* is the order of the curve. The private key, is also known as at the secret exponent *d,* is all that is required to assert ownership of a bitcoin address.

The generator G, also known as the curve domain parameters, specifies a point on the curve. Bitcoin uses the secp256k1 curve, so valid private key values range between 0x01 and 0xFFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE BAAEDCE6 AF48A03B BFD25E8C D0364141.

Public key *Q* is defined as a coordinates (*x*,*y*) calculated as multiplication of the generator G and the private exponent d; or Q = G.d

The standard construction generates an ECDSA private key and performs a SHA-256 hashing operation on the corresponding public key. This result is fed into RIPEMD-160 hash function and stored. A version byte, 0x00 for the Main Bitcoin Network, is appended to the head of the RIPEMD-160 hash and this result run through SHA-256 twice. The first 4 bytes of this hash are stored as the address checksum.

The address checksum is appended to the RIPEMD-160 hash forming a 25 byte Bitcoin Address.
For easy use by humans, this address is Base58Check encoded to a string. Base58 was developed by bitcoin to remove characters that look optically similar; 0 and O, I and l.

Assumptions and Discovery

The UTXOC construction can be applied to any *pseudonymous bitcoin-like* *cryptocurrencies*, (e.g.: Namecoin, Litecoin, Peercoin, Dogecoin, Primecoin, Bitshares-PTS).

It is suggested that the hash function used for the signature on the UTXOC match the proof-of-work hash function of the cryptocurrency. (Draft-Scrypt-RFC-2012) specifies 1.3.6.1.4.1.11591.4.11 as the OID and ASN1 syntax for parameters. Microsoft has also developed various OID's that may be useful for certain policy or certificate constraints. (MSCryptoOID 2014)

Under certain conditions, it could be applied to *semi-anonymous or 2nd generation cryptocurrencies*. (e.g.: Nxt, Ripple, Darkcoin, BlackCoin, Mastercoin, MaidSafeCoin). It is necessary that an individual transaction, and its status as being unspent, be queryable for any party at any time. Depending on the type of transaction sent within these networks, this condition may not be satisfiable.

Anonymous cryptocurrencies (e.g.: CryptoNote [Bytecoin, Monero], Darkcoin in *darksend mode*) cannot be used in this construction as it should not be possible to inspect the blockchain for an address balance without the associated private key. Clients cannot verify the *stored value* of a public key or address. Alternative constructions for binding anonymous cryptocurrency transactions could be explored at a later date.

Any reference to Bitcoin can be replaced by any other appropriate cryptocurrency without any loss of generality. Future cryptocurrencies may build this functionality into the wallet client software.

The *stored value* of a certificate is a quantitative value that corresponds to the final balance of the address associated with an address. This is calculated by inspection of the blockchain, or in practice, requested from a 3rd party blockchain API service.

This paper suggests that the binding of bitcoin to x509 be done on the transaction hash, and thus reference to a single previous payment event rather than a say the total balance associated with that private key. This satisfies the *Prior Transaction* component of the certificate, in that it could not have been generated without a embedding a previously committed transaction hash.

This allows further use cases where a certificate could be formalising a single transaction event in time for validity (the private key is spending outputs from another transaction, e.g.: a single business transaction) or the presumed secrecy of that private key (e.g.: Server Authentication keys in TLS).

It is also suggested that both a self-signed and trusted root certified model be used with this construction as each can have be applied for a different semantic purpose. (e.g: self-signed can be used to create the bearer bond type construct and trusted-root could be used within existing browsers and commercial CA trust frameworks without too much modification). Submitting a certificate signing request containing the plaintext data of a transaction and public key/address to a 3rd party permanently depseudononymizes it when signed.

Future research involving *m of n transactions* types in cryptocurrencies could provide new decentralised trust validity checks within browsers and other TLS clients. Use within other models such as GPG Web Of Trust or Certificate Convergence could be explored.

At the time of writing, Microsoft Certificate Viewer did not support the secp256k1 elliptic curve and shows UTXOC as invalid. Google Chrome TLS also did not support sec256k1 as a valid curve, returning ERR_SSL_VERSION_OR_CIPHER_MISMATCH. Firefox with NSS failed with an ssl_error_no_cypher_overlap error.

OpenSSL 1.0.1h s_client and s_server was able to negotiate a channel with a self-signed UTXOC.
OpenSSL was also able to verify a UTXOCSR, and self-sign and ca sign the CSRs.
The default openssl config does not copy SAN 2.5.29.17 so *copy_extensions = copy* should be added in the [CA_default] section.

```
openssl s_server -cert utxoc.pem
-key utxoc.key -HTTP -accept 443
-cipher ECDHE-ECDSA-AES256-GCM-
SHA384 -debug

openssl s_client -connect
localhost:443 -debug
```

It is hoped that future cryptographic libraries within browsers will support the secp256k1 curve.

Additional value – Extended Validation

Building on the concept of 'Extended Validation' (ThawteEV 2014) – a browser vendor and certificate authority UI enhancement agreement, by requiring that a CSR was signed with a private key with ownership of a certain number of coins.

Such a CA/browser vendor could require that the private key proves ownership of a number of bitcoins at a specific bitcoin address; or holds the coin in place for a certain amount of time before the certificate is signed.

In the event the private key is compromised, an attacker is more likely to transfer any explicit value associated with this address to another private key under their control.

This acts as a sign of compromise of the authentication keys to the users of a Service, which can be validated independently of the Server manually revoking the certificate.

Further, this deterrence against compromise of this key, which can be detected almost immediately by users, should promote stronger key protection, as well as more frequent key rotation by TLS Servers.

Over time, as the value of cryptocurrency fluctuates, the minimum reserve value to determine certificate validity for a particular purpose should be adjusted accordingly.

Additional Cost for an authentication forgery attack (active MiTM)

Using existing browser pinning mechanisms; (which are essentially an inverse CRLs; a list of known popular, valid certificates), a TLS service operator could assert that a Server Authentication certificate must be sufficiently expensive, that is, have a transferred a minimum balance to a certificate for its life.

In the event of a Trusted Root compromise, or an enterprise forgery CA, the substitution, forged certificates used for in the MiTM attack would need to reference a transaction that spent a sufficient amount.

This adds to the cost of the attacker, who may actually be a friendly party, such as a corporate https-inspecting Intrusion Prevention System, who must spend the same or more cryptocurrency in order to generate a valid UTXOC on their certificate substitution root. Not any old ECDSA keys signed by some authority will do, they would need to be sufficiently expensive.

This is the additional cost of storing cryptocurrency value for the lifetime of the forgery. After the forged certificate is no longer being presented to clients, its balance can be claimed and transferred to another private key under the control of the forger; so the coin is not lost.

Bearer Bonds

By reserving an address balance for a fixed period in order to maintain the validity of the UTXOC, the address owner proves that the key has not been used to spend the transaction outputs, wishing to maintain their validity for the lifetime of the certificate.

Only the holder of the private key is able to claim the output, this is the digital equivalent of the bearer bond. If the private key is compromised, it is very likely that the thief will transfer the value to another private key under their control; to prevent the owner from doing the same.

Again, this follows the same model as physical bonds; a thief that takes physical possession of the bond paper, they have essentially a right to claim its value; in practice however, its rightful owner will prevent it from being converted or transferred.

The advantage to the holder is that if the private key is compromised, there is an independent and verifiable trail showing where the value was transferred. As the cryptocurrency networks grow in maturity, it is expected that value conversion services – fiat exchanges; alt-coin exchanges, will have some way to coordinate information on fraudulent transactions and blacklist addresss sources. It may be possible in the future for honest exchanges to return stolen funds to their rightful owner, similar to how the existing regulated financial industry operates.

Additionally, it allows generation of signatures without the official client; as any PKCS#12 supporting Elliptic Curves can be used to generate these; eg: openssl

Value of compromise - Decentralised verification

In existing certificate models, the implicit value in a private key that is used for Server Authentication in TLS is that it is only known to the privileged process performing the authentication.

If this key is compromised, a passive attacker (one who can watch traffic) in the data path can decrypt previously captured TLS sessions, provided a Perfect Forward Secrecy (PFS) cipher was not used; (ie: ECDHE or DHE).

An active attacker (one who can modify traffic) can always compromise future sessions invisibily, either by subsituting the PreMasterSecret sent to the Server by the client, or decrypting the transmitted PreMasterSecret and deriving the session keys in the case of Non-PFS ciphers.

Stronger assurance against compromise can be achieved by storing keys offline or in a HSM (hardware security module, However incidents have occurred where this control has failed. (BlackTulip 2011).

As the user can lookup the balance of any bitcoin-like address through either direct inspection of the blockchain from the wallet software, or from a third party blockchain parser.

As all UTXOC are associated with at least a public address, if not an individual transaction, it is possible to inspect the balance and validity of the output from either a 3rd party blockchain service; for example https://Blockchain.info and https://chain.com

How to generate a UTXOC

Generate the secp256k1 key with

```
openssl ecparam -out transaction.key -name secp256k1 -genkey
```

This will generate an elliptic curve secret exponent (private key), defined over the curve secp256k1; the curve used by Bitcoin and derivatives.

Remove all the text above and including the line “-----END EC PARAMETERS-----“ from transaction.key

(strip curve parameters, as the curve ID is included in the EC PRIVATE KEY section)

Use the tool eckey2coin.py to display the public address of the key.
Pass the key filename with –k or paste from clipboard into standard input

```
python eckey2coin.py –k transaction.key -q transaction.png
```

This will calculate the public address associated with the EC private key. It will also generate a QR code to be used by mobile wallet software.

Initiate a transaction on the cryptocurrency network that pays this public address some amount greater than zero.

Wait a short while until the transaction is accepted by the network and been committed to a block.

From either a wallet, or a 3rd party blockchain service, find the transaction hash corresponding to your transaction. Note the block that this transaction was included in.

Use uxtocsr.py to generate a certificate singing request. Specify the hash value of the transaction that has been accepted to the network to your payment address. This can be done with the -t switch or from stdin.

```
python utxocsr.py -k transaction.key -f transaction.csr -t
transaction-hash && openssl req -in transaction.csr -text -verify
```

Once you have a CSR, this can be submitted to a Trusted Root CA for signing, or self-signed to create a new root or self-signed entity certificate.

Self Sign (make new root of validity 10 years) and verify with openssl:

```
openssl req -in transaction.csr -out transaction.crt -x509 -days
3650 -key transaction.key && openssl x509 -in transaction.crt -text
```

Verify the output of the req command generated a certificate with the included SubjectAlternativeName. You may need to tweak openssl.conf, examples are included on the github respository.

If you have built an existing CA, you can sign the CSR with your root.

openssl x509 -CA CA.crt -CAkey CA.key -req -in transacton.csr

All python code, as well as the 0.01 BTC Root Certificate should be available on the MiWCryptoCurrency github mirror:
https://www.github.com/MiWCryptoCurrency/UTXOC

You will need the following software dependancies to run these scripts:

python2, pycoin, pyasn1, pyasn_modules, qrcode

Conclusion

One can generate a UTXOC, or unspent transaction output based certificate, using an arbitrary, valid ECDSA key on the curve used by the associated cryptocurrency. By generating an x509 certificate that uses this key, one can explicitly reference the transaction in which some value was transered to this address. By ensuring that the transaction is not claimed (spent) for the lifetime of the certificate, clients have an independently verifiable way of detecting key compromise; as well as increasing the cost for active attacks against TLS.

Additionally it provides a bearer bond type construction that can be used with existing x509 infrastructure and devices such as elliptic curve smart cards; and proof of ownership signatures can be made without the official wallet software.

References

(BlackTulip 2011) 'Interim Report – Rijksoverheid.nl, Fox-It http://www.rijksoverheid.nl/bestanden/documenten-en-publicaties/rapporten/2011/09/05/diginotar-public-report-version-1/rapport-fox-it-operation-black-tulip-v1-0.pdf, 2011

(BTC2008) https://bitcoin.org/bitcoin.pdf, Satoshi Nakamoto, 2008

(BTCProto 2009) 'Protocol specification', Bitcoin Wiki, https://en.bitcoin.it/wiki/Protocol_specification

(Draft-Scrypt-RFC-2012) Scrypt KDF RFC, http://tools.ietf.org/html/draft-josefsson-scrypt-kdf-00, 2012

(HA2011) Add Honest Achmed's root certificate, Honest Achmed, https://bugzilla.mozilla.org/show_bug.cgi?id=647959, 2011

(MSCryptoOID 2014) 'Object IDs associated with Microsoft Cryptography' Microsoft, https://support.microsoft.com/kb/287547, 2014

(SCRYPT 2009) 'Stronger Key Derivation Via Sequential Memory Hard Functions', Colin Percival, *https://www.tarsnap.com/scrypt/scrypt.pdf, 2009*

(ThawteEV 2014) Extended Validation FAQ http://www.thawte.com/resources/getting-started/extended-validation-ssl-faq/