Draft, version 1

Blind signatures for Bitcoin transactions

Oleg Andreev

oleganza@gmail.com

@oleganza

Abstract. Blind signatures allow maintaining privacy while using third-party services such as digital cash server. Existing proposals of blind signatures for ECC lack compatibility with standard ECDSA and thus cannot be used directly in Bitcoin transactions. We propose a scheme that allows generating a blind signature compatible with existing Bitcoin protocol. The client requests a set of parameters from the signing server and synthesizes a public key to use in Bitcoin transaction. To redeem the funds, client transforms the hash of the transaction ("blinds"), sends to the server to sign and then transforms the signature ("unblinds") to arrive at a valid ECDSA signature. The signed transaction is published revealing the synthetic public key and the unblinded signature. Signing server cannot learn about its participation neither from the public key, nor from the signature. The scheme is particularly useful in a multi-signature transaction ("M-of-N") that securely locks funds using private keys of third parties, but keeps information about the funds absolutely private to the initiator and the recipient of the transaction.

1. Introduction

A blind signature scheme is a protocol allowing the recipient to obtain a valid signature for a message, from the signer without him or her seeing the message. Blind signature scheme is a digital signature scheme which satisfies non-forgeability and unlinkability properties. Non-Forgeability property means that only signer should be able to generate valid signatures. Every digital signature scheme should satisfy non-forgeability property. Unlinkability property means no one can derive a link between a protocol view and a valid blind signature except the requester or the author of the message. [1] The concept of blind signatures was introduced by David Chaum in 1982 [2] and extended by multiple authors in the Elliptic Curve Cryptography (ECC) [3] [4]. All works on blind signatures for ECC describe fairly simple methodology to blind messages and unblind signatures, but they all lack compatibility with existing standardized ECDSA scheme.

What is needed is a scheme that produces a compatible ECDSA signature which can be used in the current systems and protocols relying on ECDSA without any change on their part. In particular, author seeks a solution applicable to ECDSA signatures on curve *secp256k* used in Bitcoin protocol. Ability to make blind signatures directly for Bitcoin transactions would allow to

separate signing party (a "custodian") from knowing anything about the funds they are protecting. This can be viewed as an ultimate solution for secure storage of Bitcoin as any individual computer system can never be fully trusted (e.g. RNG may leak information about private keys through ECDSA signatures [5]). By spreading the trust between several independently operated computers, the risk is significantly reduced. Attacker not only has to compromise several computers instead of just one, but also has to link together their cooperation to find out which transaction to target (blind signatures make it ver). Therefore the scheme enables safer Bitcoin storage on convenient personal computers and smartphones without additional specialized hardware or compromising privacy.

2. Use case

Lets say, Alice wants to protect her bitcoins against active attackers. All her personal computing devices may be confiscated or secretly compromised in order to access her secrets. Her "paper wallets" may be stolen, or become inaccessible. Specialized "hardware wallets" can be badly compromised as well. To protect her funds, Alice may choose to lock them in a "5-of-9" multisignature transaction with 9 of her friends (possibly located in different jurisdictions, using different hardware and software). To unlock her funds, she will need any 5 of her friends to sign the redeeming transaction. Friends are instructed to authenticate Alice either in person, by phone or via other secure channel. As long as any 5 of her friends are available, did not lose their keys and no one is able to coordinate attack against the majority of the friends at the same time, her funds are much more safe than locked with just her personal keys. The only problem: Alice reveals her funds to all her friends. This reduces security drastically as some friends may attempt to conspire against Alice if she posesses a lucrative amount of bitcoins. By using blind signatures, Alice may enjoy security provided by her friends, without revealing the transaction she is signing. Assuming her friends are keeping their money safe in a similar way, all participants mutually help each other without revealing sensitive information.

3. Ordinary ECDSA signature

Let *n* be an order of the elliptic curve.

Let ${\boldsymbol{G}}$ be a standard "generator" point on the curve.

Let **t** be a private key (integer on the interval [1, n - 1]).

Let **T** be a public key corresponding to t (by definition, $T = t \cdot G$).

Let *h* be a cryptographic hash of a message to be signed (integer on the interval [1, n - 1]).

Let \boldsymbol{k} be a unique random number chosen per signature (integer on the interval [1, n - 1]).

Then the ECDSA signature is defined as a pair (*Kx*, *s*), where:

Kx is x-coordinate of the point $k \cdot G$ on the elliptic curve.

 $\mathbf{s} = k^{-1} \cdot (h + t \cdot Kx) \mod n$

Note: k^{-1} is an inverse of k modulo n such that $(k^{-1} \cdot k) = 1 \mod n$.

For brevity we will not present the verification algorithm. Suffice to say, the verification requires 3 objects to be present: message hash **h**, public key $T (=t \cdot G)$ and a signature pair (Kx, s).

4. Transformations

We observe that the core of the signature is a linear transformation of a parameter h with both factors unknown to the recipient: $s = a \cdot h + b$. We will use this idea to perform blinding of the message and unblinding of the signature.

Lets imagine Alice wants Bob to sign a Bitcoin transaction blindly. First, she needs to send him a transformed ("blinded") hash of the transaction. Then, Bob transforms ("signs") the blinded hash and returns the resulting number to Alice. Alice then transforms ("unblinds") Bob's number and arrives at a valid signature. This signature can be verified by some synthetic public key that Alice computed in advance from a combination of her secret parameters and Bob's public parameters.

Let **a**, **b**, **c** and **d** be unique random numbers within [1, n - 1] chosen by Alice.

Let **p** and **q** be unique random numbers within [1, n - 1] chosen by Bob.

Alice computes the hash of her message h and then transforms it as follows:

h₂ = *a*·*h* + *b* mod *n* (blinding)

She sends h_2 to Bob who performs another transformation:

 $\mathbf{s_1} = p \cdot h_2 + q \mod n$ (signing)

Bob sends s₁ back to Alice. She performs the final transformation:

 $\mathbf{s_2} = c \cdot \mathbf{s_1} + d \mod n$ (unblinding)

The resulting number s_2 should be a part of the signature (Kx, s_2) verifiable by a public key T (= $t \cdot G$). The only question: is it possible to determine Kx and T without compromising the secrecy of all chosen parameters? Also, T must be known in advance, before h is determined.

Lets expand s₂ as transformation of *h*:

$$s_2 = c \cdot (p \cdot (a \cdot h + b) + q) + d \mod n$$

$$s_2 = c \cdot p \cdot a \cdot h + c \cdot p \cdot b + c \cdot q + d \mod n$$
(1)

(1)

At the same time we want s_2 to be the second part of the ECDSA signature as a function of h:

$$\mathbf{s_2} = k^{-1} \cdot (h + t \cdot Kx) = k^{-1} \cdot h + k^{-1} \cdot t \cdot Kx \mod n \tag{2}$$

where

k – unique secret number within [1, n – 1]

Kx - x-coordinate of the point $k \cdot G$ (this number is the first half of the signature)

t - private key, number within [1, <math>n - 1]

We need Bob to know k and t, while Alice needs to know Kx and t \cdot G. By comparing (1) and (2) we can find the relation between ECDSA parameters with our chosen parameters a, b, c, d, p, q.

From (1) and (2) as equivalent linear transformations of an independent variable *h* follows:

$$k^{-1} = c \cdot p \cdot a \mod n \tag{3}$$

$$k^{-1} \cdot t \cdot Kx = c \cdot p \cdot b + c \cdot q + d \mod n \tag{4}$$

From (3) we can find k and K:

$$\boldsymbol{k} = (\boldsymbol{c} \cdot \boldsymbol{p} \cdot \boldsymbol{a})^{-1} \mod \boldsymbol{n} \tag{5}$$

$$\boldsymbol{K} = (\boldsymbol{c} \cdot \boldsymbol{a})^{-1} \cdot \boldsymbol{p}^{-1} \cdot \boldsymbol{G} \tag{6}$$

It's evident from (6) that Bob can communicate $p^{-1} \cdot G$ to Alice without revealing p. So Alice can know K and Kx without knowing k.

From (4) we can find t and T:

$$t = Kx^{-1} \cdot (k \cdot c \cdot p \cdot b + k \cdot c \cdot q + k \cdot d) \mod n$$

Expanding k:

$$t = Kx^{-1} \cdot ((c \cdot p \cdot a)^{-1} \cdot c \cdot p \cdot b + (c \cdot p \cdot a)^{-1} \cdot c \cdot q + (c \cdot p \cdot a)^{-1} \cdot d) \mod n$$
$$t = (a \cdot Kx)^{-1} \cdot (b + q \cdot p^{-1} + d \cdot c^{-1} \cdot p^{-1}) \mod n$$

Multiplying by *G* to arrive at a target public key:

$$T = t \cdot G = (a \cdot Kx)^{-1} \cdot (b \cdot G + (q \cdot p^{-1} \cdot G) + d \cdot c^{-1} \cdot (p^{-1} \cdot G))$$
(7)

From (7) we see that Bob can communicate EC points $(q \cdot p^{-1} \cdot G)$ and $(p^{-1} \cdot G)$ without compromising secrecy of p or q.

Now we have everything to present the protocol of constructing a blind signature by Bob for Alice.

5. Core protocol

- 1. Alice chooses random numbers a, b, c, d within [1, n 1].
- 2. Alice asks Bob to generate random numbers p, q within [1, n 1].
- 3. Bob stores his numbers p, q for later use when time comes to sign something.
- 4. Bob sends two EC points to Alice: $P = (p^{-1} \cdot G)$ and $Q = (q \cdot p^{-1} \cdot G)$.
- 5. Alice computes $K = (c \cdot a)^{-1} \cdot P$.
- 6. Alice computes public key $T = (a \cdot Kx)^{-1} \cdot (b \cdot G + Q + d \cdot c^{-1} \cdot P)$ and uses it in the output script of her transaction which locks her funds. Bob cannot know if his parameters were involved in *T* without the knowledge of *a*, *b*, *c* and *d*.

- 7. Alice sends her transaction to the Bitcoin network. Once it is included in the blockchain, transaction can only be redeemed if Bob helps Alice to produce a blind signature.
- 8. To redeem the funds, Alice creates another transaction that sends them elsewhere.
- 9. Alice computes the hash *h* of her transaction to be signed.
- 10. Alice blinds the hash and sends $h_2 = a \cdot h + b \pmod{n}$ to Bob.
- 11. Bob verifies the identity of Alice via separate communications channel.
- 12. Bob verifies that parameters p and q were never used.
- 13. Bob signs the blinded hash and returns the signature to Alice: $s_1 = p \cdot h_2 + q \pmod{n}$.
- 14. Bob marks parameters p and q as already used for the hash h_2 .
- 15. Alice unblinds the signature: $s_2 = c \cdot s_1 + d \pmod{n}$.
- 16. Alice can use (Kx, s_2) to redeem previously locked funds. This will be a valid ECDSA signature of hash *h* verifiable by public key *T*.
- 17. Every node on the network will acknowledge her signature, but she wouldn't be able to produce it alone and Bob, who helps her, will have zero knowledge about the transaction he helped to sign.

6. Security Note

Like in ordinary ECDSA, the secret parameters should never be reused in a different signature. In ECDSA, parameter *k* is either always random or at least pseudo-random, determined by the mix of the private key and the hash being signed. In our scheme, all 6 parameters must be chosen by Alice and Bob *in advance*, well before the hash *h* becomes known. While Alice has to keep track of her own pending transactions and signatures, it's not a significant burden for her to not reuse parameters for new signatures. However, Bob also must keep track of used parameters to not allow Alice (or whoever took over her computer) find them out.

In practice, the scheme only protects Alice and she trusts Bob to cooperate. So even if her computer is compromised, Bob will not sign anything unless Alice identifies herself via other means (using phone call, for example). This allows us to put more control over parameters in hands of Alice and less in Bob. Alice will still not be able to know Bob's secret numbers, but she will be able to ask Bob for a specific set of parameters matching the transaction she is about to create. As a result, Bob does not need to keep track of which parameters were already used because Alice will take care of that. Bob only needs a simple mechanism to generate required parameters sequentially on demand.

7. Generating and exchanging parameters

Alice will request parameters from Bob more than once to use in multiple transactions. In order to simplify the implementation, we propose a standard way to generate secret parameters and

exchange public parameters. This is not required for the scheme to work, but is very useful to be implemented in a standard way to have interoperable implementations.

Both Alice and Bob need to keep track of series of random parameters *per signature*. If Alice reuses the same parameters for another signature, it will allow recovery of the private key *t*. According to security note (#6), we will have Alice to keep track of the used parameters.

To generate several random numbers we will use a key derivation scheme described in BIP32 ("Hierarchical Deterministic Wallets"). In that scheme, a sequence of keys is derived from a single extended private or public key using a simple incremented index.

The operation is as follows.

- 1. Alice creates an extended private key *u*.
- 2. Bob creates an extended private key w.
- 3. Bob sends the corresponding extended public key *W* to Alice.
- 4. Alice assigns an index *i* for each "blind" public key *T* to use in a transaction. Alice must use each value of *i* only for one transaction output.
- 5. Alice generates a, b, c, d using HD(u, i), a "hardened" derivation according to BIP32:

```
a = HD(u, 4 \cdot i + 0)

b = HD(u, 4 \cdot i + 1)

c = HD(u, 4 \cdot i + 2)

d = HD(u, 4 \cdot i + 3)
```

6. Alice generates *P* and *Q* via *ND*(*W*, *i*), normal ("non-hardened") derivation according to BIP32:

 $P=ND(W,\,2{\cdot}i+0)$

$$Q = ND(W, 2 \cdot i + 1)$$

- 7. Using *a*, *b*, *c*, *d*, *P* and *Q*, Alice computes *T* and *K*.
- 8. Alice creates a transaction that uses public key *T* and publishes it on the blockchain.
- 9. Alice keeps tuple (*T*, *K*, *i*) in her wallet to be able to request and unblind the signature from Bob when redeeming the funds.
- 10. To redeem funds locked with public key *T*, Alice finds in her wallet *K* and *i*. Alice recovers her secret parameters *a*, *b*, *c*, *d* using HD(u, i).
- 11. Alice sends Bob a blinded hash $h_2 = a \cdot h + b \pmod{n}$ and index *i*.
- 12. Because *P* and *Q* are not public keys of *p* and *q*, Bob needs to do extra operations to derive *p* and *q* from *w* and *i* (following BIP32 would not be enough):

From $P = p^{-1} \cdot G = (w + x) \cdot G$ (where x is a factor in $ND(W, 2 \cdot i + 0)$) follows:

 $\boldsymbol{p} = (w + x)^{-1} \bmod n$

From $Q = q \cdot p^{-1} \cdot G = (w + y) \cdot G$ (where y is a factor in $ND(W, 2 \cdot i + 1)$) follows:

$$\boldsymbol{q} = (w+y) \cdot (w+x)^{-1} \bmod n$$

Factors *x* and *y* are produced according to BIP32 as first 32 bytes of HMAC-SHA512. See [BIP32] for details.

- 13. Bob computes blind signature $s_1 = p \cdot h_2 + q \pmod{n}$ and sends it to Alice (after verifying her indentity).
- 14. Alice receives a blind signature, unblinds it and arrives at a valid signature (Kx, s_2).

8. Conclusion

We presented a complete scheme to enable blind signatures in Bitcoin transactions. In this scheme, a signing party can provide a service of storing private keys and authenticating transactions without knowing anything about the funds being transferred. Combined with multisignature transactions, this scheme enables one to privately lock some amount of money with multiple parties.

References

[1] Kalyan Chakraborty and Jay Mehta, A Stamped Blind Signature Scheme based on Elliptic Curve Discrete Logarithm Problem (<u>http://ijns.femto.com.tw/contents/ijns-v14-n6/ijns-2012-v14-n6-p316-319.pdf</u>), 2011.

[2] David Chaum, Blind Signatures for Untraceable Payments (<u>http://www.hit.bme.hu/~buttyan/</u> <u>courses/BMEVIHIM219/2009/Chaum.BlindSigForPayment.1982.PDF</u>), 1982

[3] Chwei-Shyong Tsai, Min-Shiang Hwang, Pei-Chen Sung, Blind Signature Scheme Based on Elliptic Curve Cryptography (<u>http://mshwang.ccs.asia.edu.tw/www/myjournal/P191.pdf</u>.)

[4] Hossein Hosseini, Behnam Bahrak, Farzad Hessar, A GOST-like Blind Signature Scheme Based on Elliptic Curve Discrete Logarithm Problem (<u>http://arxiv.org/pdf/1304.2094.pdf</u>)

[5] Bruce Schneier, Did NSA Put a Secret Backdoor in New Encryption Standard? (<u>http://www.wired.com/politics/security/commentary/securitymatters/2007/11/securitymatters_1115</u>)

[BIP32] Pieter Wuille, Hierarchical Deterministic Wallets (<u>https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki</u>), 2013.