

Second draft

Blind signatures for Bitcoin transactions

Oleg Andreev

oleganza@gmail.com / @oleganza

February 22, 2014

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 the existing Bitcoin protocol. The client requests a set of parameters from the signing server and synthesizes a public key to use in a Bitcoin transaction. To redeem the funds, the 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.

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 the signer should be able to generate valid signatures. Every digital signature scheme should satisfy the 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 was extended by multiple authors to 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 current systems and protocols relying on ECDSA without any change on their part. In particular, we seek a solution applicable to ECDSA signatures used in the Bitcoin protocol. Ability to make blind signatures directly for Bitcoin transactions would allow to separate the 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 bitcoins as no individual computer system can be fully trusted (e.g. even RNG may leak information about private keys through ECDSA signatures [5]). By spreading the trust between several independently operating computers the risk is significantly reduced. The attacker not only has to compromise several computers instead of just one, but

beforehand they must find out which computers are involved (which is made much harder when blind signatures are being exchanged). Therefore the scheme enables safer Bitcoin storage on conventional personal computers and smartphones without need for specialized hardware or compromising privacy.

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 channels. As long as any 5 of her friends are available, did not lose their keys and no one is able to coordinate an attack against the majority of her friends at the same time, her funds are much safer 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 possesses a lucrative amount of bitcoins. With the help of blind signatures, Alice may enjoy security provided by her friends, without revealing transactions 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.

2. Ordinary ECDSA signature

Let n be an order of the elliptic curve.

Let G be a standard “generator” point on the curve.

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

Let k be a unique random number chosen per signature (integer in 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 modulo n .

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

Signature is invalid if either Kx or s are zero.

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

The verification requires 3 objects to be present: message hash h , public key $T (=t \cdot G)$ and a signature pair (Kx, s) . Verification steps are as follows:

1. Verify that Kx and s are integers in the interval $[1, n - 1]$.
2. Compute $w = s^{-1} \bmod n$.
3. Compute $u_1 = e \cdot w \bmod n$.

4. Compute $u_2 = Kx \cdot w \bmod n$.
5. Compute $X = u_1 \cdot G + u_2 \cdot T$.
6. If $X = \infty$ signature is invalid.
7. Signature is valid if x-coordinate of X modulo n is equal Kx .

3. 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 blind the message, signing* the blinded message and *to unblind the signature*. Note that Bob's signature will not be standard ECDSA, but after unblinding it, Alice will arrive at a valid ECDSA signature.

Lets imagine Alice wants Bob to sign a message blindly. First, she needs to send him a transformed ("blinded") hash of the message. 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:

$$h_2 = a \cdot h + b \bmod n \quad (\text{blinding})$$

She sends h_2 to Bob who performs another transformation:

$$s_1 = p \cdot h_2 + q \bmod n \quad (\text{signing})$$

Bob sends s_1 back to Alice. She performs the final transformation:

$$s_2 = c \cdot s_1 + d \bmod n \quad (\text{unblinding})$$

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 s_2 as transformation of h :

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

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

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

$$s_2 = k^{-1} \cdot (h + t \cdot Kx) = k^{-1} \cdot h + k^{-1} \cdot t \cdot Kx \bmod n \quad (2)$$

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 $[1, n - 1]$

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:

$$k^{-1} = c \cdot p \cdot a \bmod n \quad (3)$$

$$k^{-1} \cdot t \cdot Kx = c \cdot p \cdot b + c \cdot q + d \bmod n \quad (4)$$

From (3) we can find k and K :

$$k = (c \cdot p \cdot a)^{-1} \bmod n \quad (5)$$

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

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 :

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

Expanding k :

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

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

Multiplying by G to arrive at a target public key:

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

From (7) we see that Bob can communicate EC points $(p^{-1} \cdot G)$ and $(q \cdot p^{-1} \cdot G)$ without compromising secrecy of p or q . That way Alice can determine public key T that will verify her signature in the future, without Bob ever knowing T .

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

4. Core protocol

1. Alice chooses random numbers a, b, c, d within $[1, n - 1]$.
2. Bob chooses random numbers p, q within $[1, n - 1]$ and sends two EC points to Alice: $P = (p^{-1} \cdot G)$ and $Q = (q \cdot p^{-1} \cdot G)$.
3. Alice computes $K = (c \cdot a)^{-1} \cdot P$ and public key $T = (a \cdot Kx)^{-1} \cdot (b \cdot G + Q + d \cdot c^{-1} \cdot P)$. Bob cannot know if his parameters were involved in K or T without the knowledge of a, b, c and d . Thus, Alice can safely publish T (e.g. in a Bitcoin transaction that locks funds with T).
4. When time comes to sign a message (e.g. redeeming funds locked in a Bitcoin transaction), Alice computes the hash h of her message.
5. Alice blinds the hash and sends $h_2 = a \cdot h + b \pmod{n}$ to Bob.
6. Bob verifies the identity of Alice via separate communications channel.
7. Bob signs the blinded hash and returns the signature to Alice: $s_1 = p \cdot h_2 + q \pmod{n}$.
8. Alice unblinds the signature: $s_2 = c \cdot s_1 + d \pmod{n}$.
9. Now Alice has (Kx, s_2) which is a valid ECDSA signature of hash h verifiable by public key T . If she uses it in a Bitcoin transaction, she will be able to redeem her locked funds without Bob knowing which transaction he just helped to sign.

5. Security Analysis

We assume Bob may see the Alice's message (or its hash h), signature (Kx, s_2) and the public key (T) .

1. Bob does not know a and b , therefore he cannot decide if h_2 is a blinded version of h .
2. Bob does not know c and d , therefore he cannot decide if s_2 is unblinded version of s_1 .
3. Bob does not know $(c \cdot a)$, and due to difficulty of ECDLP (Elliptic Curve Discrete Logarithm Problem) cannot find them from K , therefore he cannot know if Kx was produced from his EC point P .
4. Similarly, due to difficulty of ECDLP Bob cannot learn if P and Q were used in constructing public key T .

As a result, Alice has completely hidden the fact that Bob helped her sign her message, even if she publishes everything that may need to be published (e.g. in case of Bitcoin, transaction message, signature and the public key need to be public).

Alice must not reuse parameters a and b for different messages as this would allow Bob to link blinded hashes h_2 to the original hashes h by solving two linear equations to find a and b , and then matching publicly available messages with the blinded hashes he has already signed.

Alice must not reuse c and d , as Bob can find them from the published signatures in the same way as he can find reused a and b from public and blinded hashes.

Alice cannot learn Bob's parameters p and q from P and Q (due to ECDLP), so she cannot forge his signature. However, if she makes Bob use the same parameters to sign a different message, she can calculate p and q . Bob may keep track of hashes he signed to never allow using the same pair (p, q) to sign different hash, but that's not really need. The security of the scheme depends on Bob providing secure authentication separately from Alice's secrets, so only Alice herself (not someone who stole all her secrets) can demand a valid signature from Bob. Since Bob keeps p and q only for Alice, it's not important for him if Alice tries to figure them out.

This allows us to put more control over parameters in the hands of Alice and less in Bob's. She will be able to ask Bob for a specific set of parameters matching the message she is going 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.

6. Generating and exchanging parameters

Alice will request parameters from Bob more than once to use in multiple messages. 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.

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. Each key pair is accompanied by extra 32 bytes of entropy called "chaincode" used to derive private and public keys. Since we need almost all features of BIP32 (extra entropy, sequential derivation, deriving from private and public keys, a serialization format), it would be unwise to invent similar, but incompatible scheme.

The operation is as follows.

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

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

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

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

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

5. Alice generates P and Q via $ND(W, i)$, normal (“non-hardened”) derivation according to BIP32:

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

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

6. Using a, b, c, d, P and Q , Alice computes T and K and stores tuple (T, K, i) for the future, until she needs to request a signature from Bob.

7. When Alice needs to sign her message, she retrieves K and i and recovers her secret parameters a, b, c, d using $HD(u, i)$.

8. Alice sends Bob a blinded hash $h_2 = a \cdot h + b \pmod{n}$ and index i .

9. Because P and Q are not simply the 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:

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

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:

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

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

10. Bob computes blinded signature $s_1 = p \cdot h_2 + q \pmod{n}$ and sends it to Alice (after verifying her identity).

11. Alice receives the blinded signature, unblinds it and arrives at a final signature (Kx, s_2) .

Alice does not need to interact with Bob to generate public keys. She only need to receive an extended public key W once. To request a signature matching some public key, Alice needs to tell Bob the blinded hash and the index of that public key. Bob will reply with a valid blind signature.

7. Conclusion

We presented a scheme that enables blind signatures compatible with ECDSA. The signing party can provide a service of storing private keys and signing messages, but never knowing which message they have signed. In the context of Bitcoin, it allows one party to privately lock and redeem funds (which requires publishing the public key and the signature) while the other party blindly authorizes the transaction.

The novelty of the scheme is that unlike the original Chaum blind signature scheme, this one does not allow anyone to prove that the signing party signed a particular message, but instead provides a much stronger privacy: the resulting signature, public key and the message are all completely unlinkable to the signing party. In the context of Bitcoin, the user does not need to redeem funds from a single bank (where Chaum’s blind signatures are used), but defines her own public key against which the redeeming transaction will be validated by the Bitcoin network. Essentially, her synthetic public key will act as her own private bank.

References

- [1] Kalyan Chakraborty and Jay Mehta, A Stamped Blind Signature Scheme based on Elliptic Curve Discrete Logarithm Problem (<http://ijns.femto.com.tw/contents/ijns-v14-n6/ijns-2012-v14-n6-p316-319.pdf>), 2011.
- [2] David Chaum, Blind Signatures for Untraceable Payments (<http://www.hit.bme.hu/~buttyan/courses/BMEVIHIM219/2009/Chaum.BlindSigForPayment.1982.PDF>), 1982
- [3] Chwei-Shyong Tsai, Min-Shiang Hwang, Pei-Chen Sung, Blind Signature Scheme Based on Elliptic Curve Cryptography (<http://mshwang.ccs.asia.edu.tw/www/myjournal/P191.pdf>.)
- [4] Hossein Hosseini, Behnam Bahrak, Farzad Hesar, A GOST-like Blind Signature Scheme Based on Elliptic Curve Discrete Logarithm Problem (<http://arxiv.org/pdf/1304.2094.pdf>)
- [5] Bruce Schneier, Did NSA Put a Secret Backdoor in New Encryption Standard? (http://www.wired.com/politics/security/commentary/securitymatters/2007/11/securitymatters_1115)
- [BIP32] Pieter Wuille, Hierarchical Deterministic Wallets (<https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki>), 2013.