Scriptless Scripts with ECDSA

Pedro Moreno-Sanchez, Aniket Kate

Version: April 26, 2018

Andrew Poelstra introduced the notion of Scriptless Scripts [1] as a way to realize smart contracts that no longer requires the Bitcoin scripting language. Although Bitcoin contracts have emerged as a widespread tool to build smart contracts, Bitcoin scripts have some disadvantages: First, all full nodes in the Bitcoin network must parse and validate the Bitcoin scripts apart from the signatures included in the transactions. Second, Bitcoin scripts lack the internal structure required to aggregate them. Last, but not least, Bitcoin scripts are added indeterminately in the blockchain so that they may hinder privacy and fungibility. Scriptless scripts instead, only require functionality from a digital signature scheme, which is essential already in the transactions, while the only elements visible in the blockchain are public keys and signatures.

In this state of affairs, several interesting applications of Scriptless Scripts have been proposed so far such that Mimblewimble [2], Adaptor Signatures [3] or a Scriptless version of the Lightning Network protocol [4]. All of the applications proposed so far rely on the Schnorr signature scheme. In a nutshell, they leverage the linear structure of the Schnorr signature, which ECDSA does not have. Given that, many today advocate for the inclusion of Schnorr signatures in Bitcoin as no construction for ECDSA-based Scriptless Scripts is available.

In this post, we show that it is actually possible to build Scriptless Scripts based on ECDSA. The most important advantage is that they are fully compatible with the current Bitcoin protocol as Bitcoin transactions are already signed with ECDSA signatures. Therefore, ECDSA-based constructions enable to apply Scriptless Scripts applications right now in Bitcoin!!

Preliminaries: the ECDSA signature itself

In this description, we describe the ECDSA signature scheme. Assume that Alice has a private key x, the corresponding public key g^x and she wants to sign a message m. Alice can compute a signature on a message m as follows:

1. Choose a random k
2. Compute $R := g^k$
3. Assume R is the point (r_x, r_y). Then, compute $r := r_x$. In other words, r is the x coordinate of the curve point R.
4. Compute the signature $s := k^{-1} \cdot (LSB(H(m)) + r \cdot x)$. Here LSB denotes the least significant bits function and H denotes a hash function.
5. Output (r, s)

Efficient two-party ECDSA protocol

One of the major challenges with ECDSA has been to design an efficient 2-of-2 signature protocol. Recently, Yehuda Lindell has presented a paper with an efficient construction of this protocol [5]. We sketch it here as our proposal for Scriptless Scripts is an extension of the Lindell's protocol, as we explain later. We refer the reader to the original paper to a more elaborate description and a formal security analysis.

Assume that Alice has public key $P_1 := g^{x_1}$ and nonce $R_1 := g^{r_1}$. Further assume that Bob has public key $P_2 := g^{x_2}$ and nonce $R_2 := g^{r_2}$. Finally, assume that Alice provides Bob $c_{key} := Enc_{pk_A}(x_1)$, a Paillier encryption of x_1 that only Alice can decrypt. Then, the 2-of-2 ECDSA Multisignature works as follows:

1. Alice and Bob agree on P_1, P_2, R_1, R_2 and m', where $m' = LSB(H(m))$. For security, Alice and Bob should additionally exchange non-interactive zero-knowledge to prove the knowledge of the exponents of P_1, P_2, R_1 and R_2. This is step is fundamental also in our proposed protocols in later sections. However, we omit it to improve readability.

2. Alice computes $R \leftarrow (R_2)^{r_1}$. Bob computes the $R \leftarrow (R_1)^{r_2}$. From the same R both can extract the x coordinate $r := r_x$.

3. Bob computes $c_1 \leftarrow Enc_{pk_A}((k_2)^{-1} \cdot m' + \rho q)$ and $c_2 = (c_{key})^{x_2 \cdot r \cdot (k_2)^{-1}}$. Here, ρ denotes a large random number and q denotes the modulo used during the cryptographic operations. We refer to [5] for the technical explanation why this term is required. Then, Bob computes $c_3 = c_1 \oplus c_2$, where $\oplus$ denotes the additive homomorphic operation of Paillier cryptosystem. Therefore, $c_3 = Enc_{pk_A}((k_2)^{-1} \cdot m' + \rho q + x_1 \cdot x_2 \cdot r \cdot (k_2)^{-1})$. In other words, c_3 is the encryption of a "pre-signature" from Bob. Finally, Bob sends c_3 to Alice.

4. Alice decrypts c_3 to get s'. She computes $s \leftarrow s' \cdot (k_1)^{-1}$ and output (r, s) as the ECDSA signature.

A malicious Bob here could deviate from the protocol and encrypt an arbitrary value. However, Alice can check that Bob behaved correctly by verifying that the pair (r, s) is a valid ECDSA signature under the public key Q. Otherwise, Alice can abort the protocol.

Starting the Fun: Adaptor Signatures from ECDSA

The notion of adaptor signature was introduced by Andrew Poelstra as an application of Scriptless Script to encode the following "smart contract". Assume that Bobs knows a secret value α and Alice and Bob share g^α. There are two goals that they want to achieve: (i) Alice can create a "half-signature" so that Bob can finish it only if he knows α; (ii) Alice and Bob can create a signature that reveals the value α to Alice. Poelstra showed how to realize this contract using Schnorr signatures. Here, we describe how to instantiate using ECDSA signatures.

Assume the same setting as in the Lindell's protocol. That is, assume that Alice has public key $P_1 := g^{x_1}$ and nonce $R_1 := g^{r_1}$. Further assume that Bob has public key $P_2 := g^{x_2}$ and nonce $R_2 := g^{r_2}$. Finally, assume that Alice provides Bob $c_{key} := Enc_{pk_A}(x_1)$, a Paillier encryption of x_1 that only Alice can decrypt. Additionally, assume that Bob knows α and shares g^α with Alice.

The protocol works as follows. Alice and Bob are going to create a signature that depends on three randomness (r_1, r_2, α). Bob gives Alice a "pre-signature" that only encodes his randomness r_2. Alice verifies that's actually the case and if so, she gives Bob her own "pre-signature" that encodes both r_2 and r_1. Finally, Bob constructs the final signature encoding the three randomness. We show that as soon as Bob publishes a valid signature, Alice can use it to extract α combining the published signature and her pre-signature.

In a bit more detail, the protocol works as follows:

1. Alice and Bob agree on P_1, P_2, R_1, R_2, m' and compute $Q := g^{x_1 \cdot x_2}$.

2. Bob sends $R_3 := (g^\alpha)^{r_2}$ to Alice along with a zero-knowledge proof of the fact $R_2 = g^{r_2} \wedge R_3 = (g^\alpha)^{r_2}$ without revealing the values r_2 and α.

3. Alice computes $R \leftarrow (R_3)^{r_1}$. Bob computes the $R \leftarrow (R_1)^{r_2 \cdot \alpha}$. From the same R both can extract the x coordinate $r := r_x$.

4. Bob computes $c_1 \leftarrow Enc_{pk_A}((k_2)^{-1} \cdot m' + \rho q)$ and $c_2 = (c_{key})^{x_2 \cdot r \cdot (k_2)^{-1}}$. Then, Bob computes $c_3 = c_1 \oplus c_2$, where $\oplus$ denotes the additive homomorphic operation of Paillier cryptosystem. Therefore, $c_3 = Enc_{pk_A}((k_2)^{-1} \cdot m' + \rho q + x_1 \cdot x_2 \cdot r \cdot (k_2)^{-1})$. Finally, Bob sends c_3 to Alice.

5. Alice decrypts c_3 to get s'. Here, she must verify that Bob did not cheat while performing the encryption. For that, she verifies that $(R_2)^{s' \mod q} = Q^r \cdot g^{m'}$. If it holds, she computes $s'' \leftarrow s' \cdot (k_1)^{-1}$ and send s'' to Bob.

6. Bob computes $s \leftarrow (\alpha)^{-1} \cdot s''$ and outputs the signature (r, s).

7. As soon as the signature is published, Alice computes $\alpha \leftarrow (s \cdot (s'')^{-1})^{-1}$.

Continue the Fun: Scriptless Lightning Network with ECDSA

The previous construction conceptually behaves as the hash preimage challenge that is required at the Lightning Network. The main challenge is that during the creation of the payment commitments, none of the two users Alice and Bob knows the value α to complete the signature. Only when the payment's receiver reveals α and this value reaches Bob, the signature can be finished so that coins are transferred from Alice to Bob.

Technically, the challenge appears in the step 3 of the previous protocol. In particular, if Bob does not know the value α, he cannot construct R in step 3. If we solve this issue, the rest of the protocol remains the same. In a nutshell, we solve this issue by having Alice also send her corresponding R'_3 and zero-knowledge proof that R'_3 encodes the product $r_1 \cdot \alpha$.

In the following, we show the steps for the protocol that solves the aforementioned issue:

1. Alice and Bob agree somehow on P_1, P_2, R_1, R_2, m' and $Q := g^{x_1 \cdot x_2}$.

2. Bob sends $R_3 := (g^\alpha)^{r_2}$ to Alice along with a zero-knowledge proof of the fact $R_2 = g^{r_2} \wedge R_3 = (g^\alpha)^{r_2}$ without revealing the values r_2 and α. Similarly, Alice sends $R'_3 := (g^\alpha)^{r_1}$ to Bob along with a zero-knowledge proof of the fact $R_1 = g^{r_1} \wedge R'_3 = (g^\alpha)^{r_1}$ without revealing the values r_1 and α.

3. Alice computes $R \leftarrow (R_3)^{r_1}$. Bob computes the $R \leftarrow (R'_3)^{r_2}$. From the same R both can extract the x coordinate $r := r_x$.

4. Bob computes $c_1 \leftarrow Enc_{pk_A}((k_2)^{-1} \cdot m' + \rho q)$ and $c_2 = (c_{key})^{x_2 \cdot r \cdot (k_2)^{-1}}$. Then, Bob computes $c_3 = c_1 \oplus c_2$, where $\oplus$ denotes the additive homomorphic operation of Paillier cryptosystem. Therefore, $c_3 = Enc_{pk_A}((k_2)^{-1} \cdot m' + \rho q + x_1 \cdot x_2 \cdot r \cdot (k_2)^{-1})$. Finally, Bob sends c_3 to Alice.

5. Alice decrypts c_3 to get s'. Here, she must verify that Bob did not cheat while performing the encryption. For that, she verifies that $(R_2)^{s' \mod q} = Q^r \cdot g^{m'}$. If it holds, she computes $s'' \leftarrow s' \cdot (k_1)^{-1}$ and send s'' to Bob.

6. As some time later, Bob gets the value α from the next neighbor in the payment path. Then, Bob computes $s \leftarrow (\alpha)^{-1} \cdot s''$ and outputs the signature (r, s).

7. As soon as the signature is published, Alice computes $\alpha \leftarrow (s \cdot (s'')^{-1})^{-1}$.

Multiple Possibilities

The previous construction conceptually assumes that the same α is used at each hop of a payment path. However, it is easily possible to slightly modify it so that each hop uses a different α value as the challenge pre-image.

There are a plethora of other applications that can benefit from the ECDSA-based Scriptless Scripts. For instance, the construction presented here can be leveraged to perform an Scriptless Atomic Swap compatible with current Bitcoin protocol.

Recently, other constructions for 2-of-2 ECDSA signature protocols have appeared [6]. It seems like it won't be as efficient as the one presented here.

References

[1] https://download.wpsoftware.net/bitcoin/wizardry/mw-slides/2017-03-mit-bitcoin-expo/slides.pdf
[2] https://download.wpsoftware.net/bitcoin/wizardry/mw-slides/2018-01-10-rwc/slides.pdf
[3] https://joinmarket.me/blog/blog/flipping-the-scriptless-script-on-schnorr/
[4] https://lists.launchpad.net/mimblewimble/msg00086.html
[5] https://eprint.iacr.org/2017/552.pdf
[6] https://www.computer.org/csdl/proceedings/sp/2018/4353/00/435301a595-abs.html