Schnorr Multisignatures for Bitcoin

No Author Given

No Institute Given

Abstract. Bitcoin is a digital currency system in which transactions are digitally signed by signing keys associated to all transaction inputs. It uses the ECDSA signature scheme, likely because this was the most widely-used signature scheme with the performance properties Bitcoin needed at the time of its creation. However Schnorr signatures offer more functionality than ECDSA does. In particular, Schnorr signatures support constant-size interactive multisignatures which are indistinguishable to observers from ordinary signatures. If Bitcoin supported this, it would improve both performance and user privacy.
However, while these multisignatures are algebraically simple to describe and "obvious" to implement, Bitcoin's use of ephemeral keypairs introduces new attack scenarios in which (multi-)signers are able to choose keys adversarily. This paper provides a survey of these scenarios and proposes a framework for the use of Schnorr signatures which is secure against all of them.

1 Introduction

Bitcoin[9] is a digital currency scheme in which all participants (are able to) validate transactions. These transactions[1], consist of *outputs*, which have a verification key and amount, and *inputs* which are references to outputs of earlier transactions. Each input contains a signature of the transaction with its referenced output's key.

Today, Bitcoin uses ECDSA signatures[1] over the secp256k1 curve[3] to authenticate transactions. As Bitcoin nodes fully verify all transactions, signature size and verification time are important design considerations, while signing time is much less so. Because of this, the multisignature functionality of the very similar Schnorr signature scheme seems appealing.

A complicating factor is that all keys in Bitcoin are chosen at random by the owners of outputs; any authentication of keys or association to users happens outside of the system. It is therefore important to be resilient to related-key attacks, and in the case of multisigning, adversarial key generation.

In this paper we explore the issues that arise from using Schnorr multisignatures in practice, propose a scheme that overcomes them, and show how to apply it for various use cases in cryptocurrencies like Bitcoin.

[1] This is true for all transactions except coinbase transactions which bootstrap the currency supply.

Schnorr signatures By applying the Fiat-Shamir transform[4] to the Schnorr identification scheme[11], the Schnorr signature scheme is obtained. They are defined by the equation

$$(R, s) = (kG, k - xe)$$

where $(x, P = xG)$[2] is a fixed keypair belonging to the signer, $(k, R = kG)$ is an ephemeral keypair that must be unique per-signature, and $e = H(R\|m)$ is a random oracle hash of R alongside the message m.

It can be verified by the equation

$$sG + eP = R \tag{1}$$

Since the verification equation is linear in all variables, several well-known extensions can be readily obtained.

Multisignatures Consider n signers with keypairs $\{(x_i, P_i)\}_{i=1}^n$ who wish to produce a combined signature on a message m. They act as follows.

1. They independently produce ephemeral keypairs $\{(k_i, R_i)\}_{i=1}^n$ and distribute R_i among all other parties.
2. Each computes

$$\begin{aligned} R &\leftarrow \sum_{i=1}^{n} R_i \\ e &\leftarrow H(R\|m) \\ s_i &\leftarrow k_i - x_i e \end{aligned}$$

   and distributes s_i to all other parties.
3. One party computes $s = \sum_{i=1}^n s_i$. The final signature is (R, s).

This can be verified using the above verification equation, with public key $P = \sum_{i=1}^n P_i$.

2 Pitfalls and Design Motivation

We now discuss several problems that appear when using this scheme in practical settings.

[2] Upper case variables represent elements of a group $(E, +)$ in which the discrete logarithm problem is hard. It has a generator G with order q. Lower case variables represent elements of $\mathbb{Z}/n\mathbb{Z}$. aP represents the a-times repeated application of the group operation on P.

2.1 Related public key signatures

Assuming (R, s) is a valid signature for message m and public key P, it holds that

$$sG + eP = R$$

Which implies that for every value of ν,

$$s - e\nu G + eP + e\nu G = R$$

which can be rearranged as

$$(s - e\nu)G + e(P + \nu G) = R,$$

which is again the Schnorr verification equation. We see that $(R, s - e\nu)$ is a valid signature for message m and public key $P + \nu G$. This allows third parties to obtain a valid signature for related public keys, without access to the corresponding private keys. Note that since this is not possible for chosen public keys, this does not break the security of Schnorr signatures[12]. However, in protocols where the public keys are not fixed in advance, this may be problematic.

This is particular worrisome in combination with Bitcoin's BIP32[15] key derivation, where such related keys may be used

$$P_{\text{child}_i} = P_{\text{parent}} + f(P_{\text{parent}}, i)G$$

where f is the BIP32 key derivation function. This would allow any third party who observes a signature (R, s) for P_{child_i} to convert it to another key P_{child_j} by replacing it with

$$(R, s + e[f(P_{\text{parent}}, i) - f(P_{\text{parent}}, j)])$$

while only knowing the public key P_{parent}.

Solution The standard solution[2] for this issue is to include the verification public key in the hash, and define

$$e \leftarrow H(R \| P \| m)$$

without modifying any of the other equations.

2.2 Derandomized signing

To avoid the need for a strong random number generation at signing time, the creation of the ephemeral keys k_i is often done using a algorithm like RFC6979[10], which computes them using a deterministic function $f(x_i, m)$. When multiple signers are cooperating, one must ensure that the same ephemeral key is not reused when other participants change their ephemeral keys in a repeated signing attempt. Otherwise participants can recover others' private keys.

Example attack Alice and Bob want to jointly produce a signature. Alice produces an ephemeral keypair (k_A, R_A) and sends R_A to Bob. In a first attempt, Bob responds with a point $R_B^{(1)}$. Alice computes

$$\begin{aligned} R^{(1)} &\leftarrow R_A + R_B^{(1)} \\ e^{(1)} &\leftarrow H(R^{(1)} \| P \| m) \\ s_A^{(1)} &\leftarrow k_A - x_A e^{(1)} \end{aligned}$$

and sends $s_A^{(1)}$ over to Bob. Bob chooses not to produce a valid $s_B^{(1)}$, and thus subsequent protocol steps fail. A new signing attempt takes place, and Alice again sends R_A. Bob responds with a point $R_B^{(2)} \neq R_B^{(1)}$. Alice computes

$$\begin{aligned} R^{(2)} &\leftarrow R_A + R_B^{(2)} \\ e^{(2)} &\leftarrow H(R^{(2)} \| P \| m) \\ s_A^{(2)} &\leftarrow k_A - x_A e^{(2)} \end{aligned}$$

and sends $s_A^{(2)}$ over. Bob can now derive

$$x_A = \frac{s_A^{(1)} - s_A^{(2)}}{e^{(2)} - e^{(1)}}$$

Solution To avoid this problem, we must ensure that whenever any R_i point (or the message m) changes, the k_i values change. As long as f is deterministic, this implies a circular dependency in the choice of ephemeral keys. It can be solved by introducing (non-repeating) randomness or a counter into the function f. Unfortunately, this requires a secure random number generator at signing time, or state that is kept between signing attempts.

2.3 Public key cancellation

Motivating example Alice and Bob want to jointly produce a signature. Alice has keypair (x_A, P_A), and Bob has (x_B, P_B). However, Bob instead lists his public key as $P'_B = P_B - P_A$. In this case, verifiers will expect a valid signature for the combined public key

$$P = P_A + P'_B = P_A + P_B - P_A = P_B$$

which Bob can produce without access to Alice's private key.

In general, whenever some signer can choose their public key after observing some other signers' keys, it is possible for them to choose in such a way that the others' keys are cancelled out.

Traditionally, one would expect all public keys to be chosen and certified in advance. This is possible when there is a certificate authority in play, or when all verifiers demand a signature by all signers' claimed public keys.

Pointwise delinearization Here we explore an alternate solution that does not require out-of-band validation. We change the computation of s_i in the original multisigning scheme to

$$s_i \leftarrow k_i - H(P_i)x_i e$$

and the resulting combined public key to

$$P = \sum_{i=1}^{n} H(P_i)P_i$$

This delinearization makes it impossible for any single signer to choose a public key in such a way that they can learn the private key corresponding to the combined key. We make this more precise in the next theorem.

Theorem 1. *Suppose an attacker is given a combined public key P and can produce Q and x such that*

$$P + H(Q)Q = xG$$

where H is a random oracle. This attacker can be used to directly produce Schnorr signature forgeries with key P.

Proof. Choose a message m to forge on, and give the attacker a challenge key as P. For all random oracle queries X, reply with $\frac{-1}{H'(X\|P\|m)}$, where H' is the random oracle used by the Schnorr signature scheme. Let the attacker produce x and Q as in the theorem statement. Then

$$P - H(Q)Q = P - \frac{Q}{H'(Q\|P\|m)} = xG$$

This can be rearranged as

$$-xH'(Q\|P\|m)G + H'(Q\|P\|m)P = Q,$$

which is the Schnorr verification equation. We conclude that $(Q, -xH'(Q\|P\|m))$ is a valid Schnorr signature for message m and public key P.

We observe that this theorem exactly covers the case when an attacker controls a single public key: he waits for the other signers to reveal their keys, and computes

$$P = \sum_{i=1}^{n-1} H(P_i)P_i$$

and reveals his own key as Q. Unfortunately, the delinearization scheme fails to protect against the case where multiple signers can collaborate to cancel out the keys of others.

Multiple key cancellation attack Assume the honest participants control m out of n keys, and their keys combine to

$$P_{\mathrm{h}} = \sum_{i=1}^{m} H(P_i)P_i$$

The attackers now choose a large number of secret numbers, and use *Wagner's algorithm*[14] to choose $\{r_i\}_{i=m+1}^{n}$ out of them such that

$$\sum_{i=m+1}^{n} H(P_{\mathrm{h}} + r_i G) = -1$$

modulo the order of the group. They then reveal their public keys $\{P_i\}_{i=m+1}^{n}$ as $\{P_{\mathrm{h}} + r_i G\}_{i=m+1}^{n}$. The combined key then becomes

$$P_{\mathrm{h}} + \sum_{i=m+1}^{n} H(P_{\mathrm{h}} + r_i G)(P_{\mathrm{h}} + r_i G)$$

which equals

$$\left(1 + \sum_{i=m+1}^{n} H(P_{\mathrm{h}} + r_i G)\right) P_{\mathrm{h}} + \left(\sum_{i=m+1}^{n} r_i H(P_{\mathrm{h}} + r_i G)\right) G$$

which, due to the choice of $\{r_i\}_{i=m+1}^{n}$, cancels out to

$$\left(\sum_{i=m+1}^{n} rH(P_{\mathrm{h}} + r_i G)\right) G$$

As a result, the attackers can produce a valid signature with as private key

$$\sum_{i=m+1}^{n} r_i H(P_{\mathrm{h}} + r_i G)$$

Full delinearization The correct solution puts all signers' public keys under the hash to compute delinearization factors. We define a *key set commitment* $C = H(P_1 \| P_2 \| \dots \| P_n)$, and change the computation of s_i to

$$s_i \leftarrow k_i - H(C \| P_i) x_i e$$

and the resulting combined public key to

$$P = \sum_{i=1}^{n} H(C \| P_i) P_i \tag{2}$$

We can generalize our claim from the previous section to allow the attacker to produce multiple public keys — or equivalently, to allow multiple signers to collude.

Theorem 2. *Let $\mathcal{A}$ be an algorithm which receives as input m public keys $\{P_i\}_{i=m+1}^n$, outputs public keys $\{P_i\}_{i=m+1}^n$ and secret key x such that*

$$\sum_{i=1}^{n} H'(C\|P_i)P_i = xG$$

where H' is a random oracle and $C = H'(P_1\| \cdots \|P_n)$ as above.

Then a random oracle simulator for $\mathcal{A}$ exists which can solve the discrete logarithm problem.

Proof. The simulator acts as follows. It begins with a discrete logarithm challenge (G, Q) where its goal is to determine y such that $Q = yG$. It then acts as follows.

1. It chooses $m-1$ keypairs (y_i, P_i) for $i = 2, \ldots, m$. It sets $P_1 = Q$ and gives $\{P_i\}_{i=1}^m$ to $\mathcal{A}$.
2. It answers random oracle queries uniformly at random, except when it receives a query of the form $C\|P_1$, where C is itself the output of an earlier query. In this case, it forks $\mathcal{A}$ and replies with uniformly random numbers h_ℓ, h_r to the respective forks.
   (We assume $h_\ell \neq h_r$, which is true except with negligible probability, and we can assume without loss that such a situation only happens once, since only one query of the form $C\|P_1$ will contain information correlated with $\mathcal{A}$'s output.)
3. Eventually, the respective forks of $\mathcal{A}$ output $n-m$ public keys $\{P_i\}_{i=m+1}^n$ and respective secret keys x_ℓ and x_r such that

$$h_\ell P_1 + \sum_{i=2}^{n} H(C\|P_i)P_i = x_\ell G$$

$$h_r P_1 + \sum_{i=2}^{n} H(C\|P_i)P_i = x_r G$$

   (Note that both forks must output the same public keys, since the fork occurred after they were chosen, except with negligible probability. This is because the forking query contained the variable C that $\mathcal{A}$ could not know before choosing the public keys.)

It is then immediate that

$$Q = P_1 = \frac{x_\ell - x_r}{h_\ell - h_r} G$$

Thus the discrete logarithm challenge is solved.

We observe that in the special case $n = 2$, we can rearrange the commitments to obtain the situation from the previous section. We proved this secure by showing that single key cancellation was equivalent to producing a Schnorr signature. On the other hand, we can use this fact, combined with the above proof, to obtain a proof that Schnorr signatures are one-time secure.

3 Applications

We now describe how the resulting scheme can be applied to cryptocurrencies in various ways.

3.1 `OP_CHECKSIG` replacement

The most obvious application is as a direct replacement of the `OP_CHECKSIG` transaction Script opcode. In its current form in Bitcoin, it acts as an operator that pops a public key and an ECDSA signature off the stack, and verifies the signature against the key with the current transaction as message. In a new Script version, this opcode could be replaced with an equivalent one that pops a public key P and a Schnorr signature (R, s) off the stack instead, and verifies the transaction m against it using Equation 1.

The immediate advantage is that whenever multiple signatures are requested for authorizing a spend, a simple `OP_CHECKSIG` opcode can be used instead of the costlier and less private `OP_CHECKMULTISIG` to encode an n-of-n multisig policy. The various signers simply act a single one towards the public, using the combined public key according to Equation 2. To sign, they interact internally until a single combined signature is obtained, and present it to the network afterwards.

Furthermore, this would be an opportunity to improve some long-standing issues with malleability and inefficiency of DER signatures [16] that are currently used.

3.2 `OP_CHECKMULTISIG` replacement

The above approach is unfortunately limited to cases where signatures from all signers are needed. Bitcoin currently supports simple threshold policies where only m-of-n signers are needed to authorize a transaction, using the `OP_CHECKMULTISIG` opcode. It pops a list of public keys, a list of corresponding signatures, and values for m and n from the stack. This is inefficient, as its size and validation time scale linearly with the number of participants — but it is more flexible than what we can do with our multisignature scheme[3].

We can, however, provide a replacement for `OP_CHECKMULTISIG` as well. Instead of popping a list of signatures, it pops just one combined signature and a list of which keys are to be used. Validators compute the combined public key from the chosen subset of public key, though the delinearization step $(P_i \mapsto H(C\|P_i)P_i)$ can be done ahead of time by the signers — it exists purely to protect the signers from each other.

[3] There exists a threshold scheme for Schnorr signatures [13], but it requires the participants to interact to establish the keys, rather than being able to choose them independently. This is compatible with our proposal, and can be used still to distribute its keys to more participants.

3.3 Signature aggregation across inputs

The previous two applications generally reduce the number of signatures to verify per transaction input to 1, but we would like to go further, and aggregate all signatures from all transaction inputs into 1. Here we do not have a prepublished commitment to the set of signers, as each transaction input can spend an output that requires authorization from distinct participants. We do not wish to restrict this functionality, as it would interfere with fungibility improvements like CoinJoin[6].

This is where delinearization comes in. If the validation were to simply add public keys like in Section 3.2, an attacker can steal any coins using the cancellation attacks from Section 2.3. To do so, he proceeds as follows:

1. The attacker chooses any number of transaction outputs $\{O_i\}_{i=1}^{n-1}$ he wants to steal, and identifies their public keys $\{P_i\}_{i=1}^{n-1}$.
2. He computes the public key $P_n \leftarrow xG - \sum_{i=1}^{n-1}$, where x_n is his own private key.
3. He produces a transaction that sends a small amount of his own money to create an output O_n with public key P_n.
4. He can now construct a transaction that spends $\{O_i\}_{i=1}^{n}$, while signing with private key x.

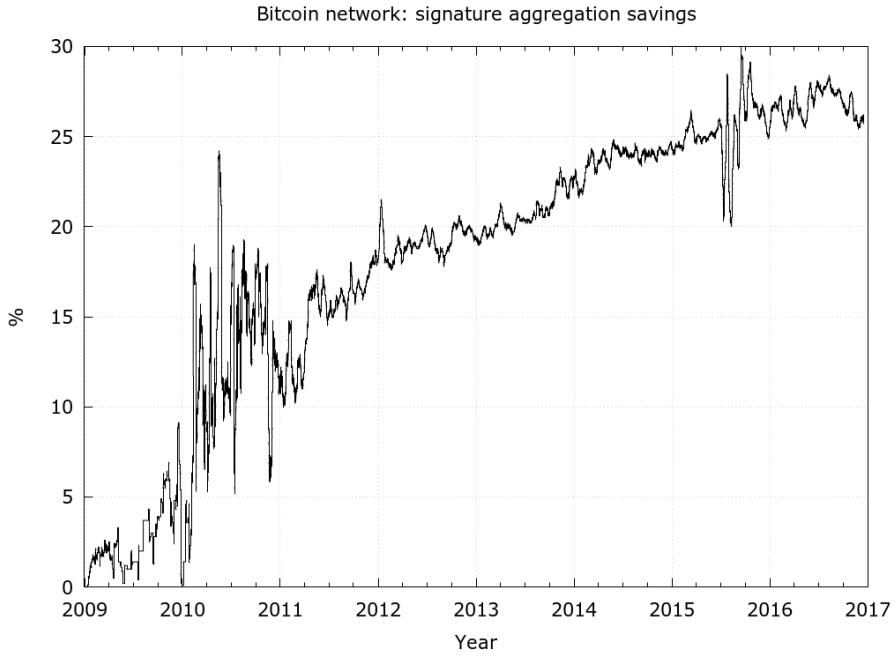
Even pointwise delinearization would be attackable. Especially as an attacker can easily choose to add hundreds of extra transaction inputs, Wagner's algorithm would very efficiently let him cancel that out. However, when the verifier fully delinearizes the public keys before combining, this is not possible.

Implementation To implement this, we can propose that all `OP_CHECKSIG` and `OP_CHECKMULTISIG`-like opcodes be modified to take 1-byte arguments that indicate whether or not a combined signature for that position is present, instead of actual signatures. The opcodes retain their policy check — for example, a 2-of-3 multisig check will still verify that 2 of the 3 signatures indicate presence. However, they also accumulate a list of all public keys for which signatures are claimed to be present. Finally, after individual transaction input scripts are evaluated, a single transaction-wide signature is verified against the delinearized combination of all public keys in the accumulator. To sign for this, all transaction input owners (typically only 1) collaborate to produce the single aggregated signature.

Performance As this scheme requires delinearization at validation time, it no longer runs in (nearly) constant time. However, the public key combination equation and the validation equation can be combined into one. Its performance is similar to that of batch validation[8].

Analysis We analysed the Bitcoin blockchain to determine the potential storage and bandwidth savings that would result from the scheme above. We assume every signature would be reduced to 1 byte, except for one per transaction. This

does not account for the potential savings from n-of-n thresholds that could be reduced to single signatures. Overall, a 25% reduction is observed, with more recent blocks showing a slightly higher reduction.



4 Future work

4.1 Merkle key trees

Instead of publishing all participant public keys and letting the verifier combine them like in Section 3.2, we can instead compute all valid subsets and combine them ahead of time — essentially rewriting the m-of-n policy as a 1-of-n.

This quickly becomes unwieldy when $\binom{n}{m}$ grows too large. Instead of publishing them, a Merkle tree[7] over the combined keys can be computed, and only its root needs to be revealed. The signers then pick one combination, produce a combined signature for it, and reveal the combined key, signature, and Merkle branch to prove its inclusion in the tree. This scheme only needs $\mathcal{O}(\log_2 \binom{n}{m})$ space — often less than the $\mathcal{O}(n)$ needed by `OP_CHECKMULTISIG`.

4.2 Polynomial-key multisignature

The tree-based scheme in Section 4.1 is not computationally feasible when the number of possible satisfactions becomes too large, and the `OP_CHECKMULTISIG` scheme in Section 3.2 is inefficient when the number of public keys involved

is large. Since threshold signatures are usually used in Bitcoin for redundancy reasons, it would be attractive to construct a scheme which is more efficient in the common case where more keys participate.

Example Consider the scheme from Section 3.2 with public keys $\{P_i\}_{i=1}^{5}$, where a threshold of 4 signers is wanted. Instead of publishing $\{P_i\}_{i=1}^{5}$, consider publishing only the two points $P_{C0} = \sum_{i=1}^{5} P_i$ and $P_{C1} = \sum_{i=1}^{5} iP_i$. Then if, for example a signature with P_1, P_2, P_3, and P_5 is wanted, the following combination key is used:

$$P_{1235} = 4P_{C0} - P_{C1}$$

It is equal to

$$P_{1235} = \sum_{i=1}^{5} (4 - i)P_i$$

which simplifies to

$$P_{1235} = 3P_1 + 2P_2 + P_3 - P_5$$

which can be signed for without knowing x_4. By choosing different combinations of P_{C0} and P_{C1}, any single participant key can be cancelled out at verification time.

This scheme can be generalized. By publishing $1 + n - m$ polynomial combinations of the participant keys, any m-of-n threshold policy can be implemented.

5 Conclusion

We have described a number of the practical challenges that arise when applying Schnorr multisignatures to the Bitcoin system, and suggested solutions to these challenges which also provide a number of additional benefits

If deployed, our proposed work would reduce the going-forward storage and bandwidth requirements of the Bitcoin system by approximately 25% while improving user privacy and without introducing any new cryptographic strong assumptions or undermining accountability.

These improvements can be deployed in a disruption free and fully backwards compatible mechanism by utilizing the script versioning mechanism that is part of the Segregated Witness upgrade[5] making their deployment a realistic prospect in the immediate future.

References

1. ANSI Standards Committee Financial Services: Public Key Cryptography for the Financial Services Industry - the Elliptic Curve Digital Signature Algorithm (ECDSA): ANSI American National Standard for Financial Services, ANS X9.62-2005. Accredited Standards Committee X9, Incorporated (2005)
2. Bernstein, D.J.: Multi-user schnorr security, revisited. Cryptology ePrint Archive, Report 2015/996 (2015), `http://eprint.iacr.org/2015/996`

3. Certicom Research: SEC 2: Recommended elliptic curve domain parameters. http://www.secg.org/SEC2-Ver-1.0.pdf (2000)
4. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: CRYPTO. pp. 186–194 (1986)
5. Lombrozo, E., Lau, J., Wuille, P.: BIP141: Segregated witness (consensus layer). Bitcoin Improvement Proposal (2015), https://github.com/bitcoin/bips/blob/master/bip-0141.mediawiki
6. Maxwell, G.: CoinJoin: Bitcoin privacy for the real world (2013), BitcoinTalk post, https://bitcointalk.org/index.php?topic=279249.0
7. Merkle, R.: A digital signature based on a conventional encryption function. In: Lecture Notes in Computer Science, vol. 293, p. 369 (1988)
8. Naccache, D., Vaudenay, S., Raphaeli, D., cole Normale Suprieure: Can d.s.a. be improved? - complexity trade-offs with the digital signature standard. pp. 77–85. Springer-Verlag (1995)
9. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system (2009), https://www.bitcoin.org/bitcoin.pdf
10. Pornin, T.: Deterministic Usage of the Digital Signature Algorithm (DSA) and Elliptic Curve Digital Signature Algorithm (ECDSA). RFC 6979 (Aug 2013), https://rfc-editor.org/rfc/rfc6979.txt
11. Schnorr, C.P.: Efficient Identification and Signatures for Smart Cards, pp. 688–689. Springer Berlin Heidelberg (1990)
12. Seurin, Y.: On the Exact Security of Schnorr-Type Signatures in the Random Oracle Model, pp. 554–571. Springer Berlin Heidelberg (2012)
13. Stinson, D.R., Strobl, R.: Provably secure distributed schnorr signatures and a (t, n) threshold scheme for implicit certificates. In: Proceedings of the 6th Australasian Conference on Information Security and Privacy. pp. 417–434. ACISP '01, Springer-Verlag (2001)
14. Wagner, D.: A generalized birthday problem. In: In CRYPTO. pp. 288–303. Springer-Verlag (2002)
15. Wuille, P.: BIP32: Hierarchical deterministic wallets. Bitcoin Improvement Proposal (2013), https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki
16. Wuille, P.: BIP62: Dealing with malleability. Bitcoin Improvement Proposal (2014), https://github.com/bitcoin/bips/blob/master/bip-0062.mediawiki