Int. J. Inf. Secur. (2017) 16:213–226
DOI 10.1007/s10207-016-0325-1

REGULAR CONTRIBUTION

Two-factor authentication for the Bitcoin protocol

Christopher Mann[1] · Daniel Loebenberger[1]

Published online: 5 April 2016

Abstract We show how to realize two-factor authentication for a Bitcoin wallet. To do so, we explain how to employ an ECDSA adaption of the two-party signature protocol by MacKenzie and Reiter (Int J Inf Secur 2(3–4):218–239, 2004. doi:10.1007/s10207-004-0041-0) in the context of Bitcoin and present a prototypic implementation of a Bitcoin wallet that offers both: two-factor authentication and verification over a separate channel. Since we use a smart phone as the second authentication factor, our solution can be used with hardware already available to most users and the user experience is quite similar to the existing online banking authentication methods.

Keywords Bitcoin · Two-party ECDSA · Two-factor authentication · Block chain

CR Subject Classification Security and privacy · Digital signatures · Mobile and wireless security

1 Introduction

Bitcoin (BTC) is a cryptographic currency proposed by Satoshi Nakamoto [23] in the legendary e-mail to the Cryptography Mailing list at `metzdowd.com`. One of the most important features of Bitcoin is that it is completely peer-to-peer, i.e., it does not rely on a trusted authority (the bank) which ensures that the two central requirements of any electronic cash system are met: Only the owner can spend money and it is impossible to spend money twice. In Bitcoin, these two features are realized with a common transaction history, the Bitcoin *blockchain*, known to all users. Each of the transactions in the chain contains the address to which some Bitcoins should be payed, the address from which the Bitcoins should be withdrawn and the amount. Both addresses are directly derived from the public key of the corresponding ECDSA key pairs of the recipient and the sender, respectively. The whole transaction is then signed using the ECDSA private key of the sender. Since any user might have multiple addresses, its *wallet* consists of several key pairs and is typically stored on the owner's device or within some online service.

Thus, from a thieves' perspective, the only thing one has to do in order to steal some Bitcoins is to get one's hands on the corresponding wallet, just like in real life. Indeed, Lipovsky [19] describes an online banking Trojan that also steals Bitcoin wallets.

A common approach to complicate this is the use of two-factor authentication. This means that the wallet stored on a device does *not* contain the private keys but just shares of them. The other shares are stored on an independent device (such as a smart phone). Now, any transaction can only be signed with the help of *both* shares of the private key. During the signing process, it has to be ensured that at no point in time the full private key is present on either of the devices.

There was already considerable effort to realize two-factor authentication for Bitcoin wallets. First of all, it is in principle possible to use Bitcoin's built-in functionality for threshold signatures, see Sect. 2. This has, however, three major disadvantages: First of all, it would be visible in the blockchain that multi-factor authentication is used. Second, the size of the transaction increases, which leads to higher transaction fees. Last but not least, there are Bitcoin clients around which do not work properly with the threshold signature extension.

✉ Daniel Loebenberger
daniel@bit.uni-bonn.de

Christopher Mann
christophermann@web.de

[1] B-IT, University of Bonn, Bonn, Germany

Goldfeder, Bonneau, Felten, Kroll, and Narayanan [11] tried to employ threshold signatures proposed by Ibrahim, Ali, Ibrahim, and El-sawi [14]. However, as the authors pointed out there, it is quite difficult to use these kind of signatures for two-factor authentication, since the restrictions on the threshold are quite delicate to handle. In their blog post, they compare different threshold signatures with respect to their applicability to Bitcoin wallets. However, their reasoning remains quite high level.

In this article, we show in Sect. 3 how to actually realize two-factor authentication for a Bitcoin wallet employing the two-party ECDSA signature protocol adapted from MacKenzie and Reiter [20]. In Sect. 4, we also present a prototypic implementation of a Bitcoin wallet that offers both: two-factor authentication and verification over a separate channel. Since we use a smart phone as the second authentication factor, our solution can be used with hardware already available to most users and the user experience is quite similar to the existing online banking authentication methods. Our source code is liberally licensed and can be found on GitHub, see Mann [21]. We also got in contact with the developers of the Java Bitcoin library. Indeed, there was lively discussion on the Bitcoin mailing list, when they got aware of our prototype. For details, see Hearn [13]. A preliminary version of this article was published in Mann and Loebenberger [22].

Very recently, we found the work of Goldfeder, Gennaro, Kalodner, Bonneau, Kroll, Felten, and Narayanan [12], where the authors present an extended version of the MacKenzie and Reiter scheme which allows t-party threshold signatures. This is a very nice idea in the context of Bitcoin, and it would be very interesting to see their extended scheme running. Unfortunately, their prototype currently only implements the plain MacKenzie and Reiter scheme. Furthermore, we observed that in contrast to our implementation their desktop wallet serves as a trusted dealer during initialization. On a compromised computer, this is a clear security problem. We address this issue here, see Sect. 3.4.2.

2 Threshold signature support in Bitcoin

As part of the scripting functionality, Bitcoin supports t-out-of-u threshold signatures. Instead of only a single signature, a user must provide t signatures to spend a transaction output. Each of the t signatures must verify under one of the u public keys. Bitcoin's threshold signature support has been used by Bitpay Inc. [3] to implement a Web application that offers shared control of Bitcoin addresses.

In the standard single signature case, Bitcoins are sent to a Bitcoin address which is directly derived from a public key. The payee can spend the received Bitcoins by providing a transaction with a signature that verifies under the public key. In the threshold signature case, the payer must specify a list of u public keys instead of a single one. The payee can spend the received Bitcoins by providing a transaction with t signatures where each of the signatures verifies under one of the u public keys.

As a list of public keys is now used to identify the payee instead of a single one, no Bitcoin address can be derived any more. Thus, the payer must not only know a short Bitcoin address but the whole list of u public keys to send Bitcoins to the payee. This is very inconvenient for the payer. A further Bitcoin feature called pay-to-script-hash (P2SH) solves this problem by adding another indirection: Instead of specifying the whole list of public keys, the payer only specifies the hash value of a Bitcoin script, which contains the list of public keys. The script is hashed with the same function that is used to hash the public keys. Therefore, it is possible to derive a Bitcoin address from the script. When spending the Bitcoins, the payee must not only provide the t signatures, but also a Bitcoin script that fits the hash value specified by the payer. The signatures in the spending transaction are then verified against the public keys in the script.

The combination of both features provides a threshold signature support that is as convenient for the payer as the single signature version of Bitcoin. Nevertheless, this variant of threshold signatures for Bitcoin has the three disadvantages mentioned above: First, it is visible in the public block chain that threshold signatures are used. This implies that everybody can see by inspection of the block chain that a certain amount of Bitcoins is shared by different addresses. Even though this does not directly reveal the use of two-factor authentication, it still breaks the semantics of a Bitcoin address, since we do not want to consider the phone as a second person owning the Bitcoins, but just as a mandatory authentication method for a user, who wants to spend them. Second, the spending transaction becomes much larger as it contains the t signatures and the script with the list of the u public keys. Signatures and public keys are responsible for most of the data in a transaction. Consequently, having several of them increases the size of the transaction significantly and can increase the transaction fees as these depend on the size of the transaction. Last but not least, there are Bitcoin clients around which do not work properly with the threshold signature extension. The use of threshold signatures compatible with ECDSA as discussed in the next section circumvents these kinds of problems.

3 Two-party ECDSA

In MacKenzie and Reiter [20], a two-party signature scheme for DSA is presented. It employs a homomorphic cipher, specifically the Paillier [25] cryptosystem. This allows one party to operate with cipher texts of another party's secrets without ever learning about these secrets. In difference to

$$
\begin{array}{lcl}
\textbf{Alice} & & \textbf{Bob} \\
d_A \xleftarrow{R} \mathbb{Z}_n^{\times} & & \\
Q_A \longleftarrow d_A G & & \\
(\mathrm{sk}_A, \mathrm{pk}_A) \longleftarrow \mathrm{Gen}_{\mathrm{Pai}}() & & \\
(\mathfrak{g}_B, \mathfrak{h}_B, \mathfrak{N}_B) \longleftarrow \mathrm{Gen}_{\mathrm{IC}}() & & \\
\tau_A \longleftarrow (\mathrm{p}_{\mathrm{init}}(\mathfrak{g}_B), \mathrm{p}_{\mathrm{init}}(\mathfrak{h}_B)) & \xrightarrow[(\mathfrak{g}_B, \mathfrak{h}_B, \mathfrak{N}_B)]{Q_A, \mathrm{pk}_A, \tau_A} & \text{check } \tau_A, Q_A \in \langle G \rangle \\
 & & \text{check } \neg\,\mathrm{prime}(\mathfrak{N}_B) \\
 & & d_B \xleftarrow{R} \mathbb{Z}_n^{\times} \\
 & & Q_B \longleftarrow d_B G \\
 & & (\mathrm{sk}_B, \mathrm{pk}_B) \longleftarrow \mathrm{Gen}_{\mathrm{Pai}}() \\
 & & (\mathfrak{g}_A, \mathfrak{h}_A, \mathfrak{N}_A) \longleftarrow \mathrm{Gen}_{\mathrm{IC}}() \\
\text{check } \tau_B, Q_B \in \langle G \rangle & \xleftarrow[(\mathfrak{g}_A, \mathfrak{h}_A, \mathfrak{N}_A)]{Q_B, \mathrm{pk}_B, \tau_B} & \tau_B \longleftarrow (\mathrm{p}_{\mathrm{init}}(\mathfrak{g}_A), \mathrm{p}_{\mathrm{init}}(\mathfrak{h}_A)) \\
\text{check } \neg\,\mathrm{prime}(\mathfrak{N}_A) & & \\
Q \longleftarrow d_A Q_B & & Q \longleftarrow d_B Q_A
\end{array}
$$

Fig. 1 Pairing protocol for performing the setup for the two-party ECDSA signature protocol in Fig. 2. $\mathrm{Gen}_{\mathrm{Pai}}()$ executes the setup for the Paillier cryptosystem and $\mathrm{Gen}_{\mathrm{IC}}()$ executes the setup for the Fujisaki and Okamoto integer commitment scheme as described in Sect. 3.1. $\mathrm{p}_{\mathrm{init}}(x)$ generates a (non-zero-knowledge) proof that x is a quadratic residue modulo $\mathfrak{N}_A$ or modulo $\mathfrak{N}_B$

the other threshold signature schemes, this one works for only two parties. As we need a two-party signature scheme for ECDSA to implement our two-factor wallet, we decided to port their scheme to ECDSA. Also Goldfeder et al. [11] came to the same conclusion: In the blog post related to their article, they note that the scheme by MacKenzie and Reiter seems to be "close to ideal." They later describe in Goldfeder et al. [12] a t-out-of-n extension for the scheme proposed by MacKenzie and Reiter which uses a threshold version of the Paillier cryptosystem. Note that there are also other applications of multi-party signatures in the context of Bitcoin such as coin mixing for improving the privacy, see Ziegeldorf, Grossmann, Henze, Inden, and Wehrle [32].

We now give an overview of two-party signatures as described by MacKenzie and Reiter [20] in the context of ECDSA.

For the setup, one fixes a cryptographic hash function h (in our case, we use SHA-256, see NIST [24]) and a particular set of elliptic curve domain parameters: a prime power $q \in \mathbb{N}_{\geq 2}$ denoting the size of the base field, the elliptic curve parameters $a, b \in \mathbb{F}_q$ defining the elliptic curve $E\colon y^2 = x^3 + ax + b$, a (finite) base-point $G \in E$ of prime order $n \in \mathbb{N}$, and a cofactor $h = \#E/n \in \mathbb{N}$. An *ECDSA key pair* is a pair $(d, Q) \in \mathbb{Z}_n^{\times} \times E$, where d was pseudorandomly generated and $Q = dG$ on the elliptic curve E. In the case of Bitcoin, the elliptic curve `secp256k1` as defined by Certicom Research [6] is fixed. There, q is a large prime, $a = 0$, $b = 7$ and the cofactor is $h = 1$. In order to sign a message $m \in \{0, 1\}^*$ in ECDSA, Alice selects pseudorandomly a nonzero integer $k \in \mathbb{Z}_n^{\times}$ and computes kG. The process is repeated as long as the x-coordinate $r = \mathrm{coord}_x(kG) \bmod n = 0$. Now, Alice computes $s = k^{-1}(\mathrm{h}(m) + rd)$. If $s = 0$, the process is repeated using a new ephemeral key $k \in \mathbb{Z}_n^{\times}$.

The ECDSA version of the signature scheme by MacKenzie and Reiter [20] consists of three different phases for jointly signing a message $m \in \{0, 1\}^*$.

1. Initialization. In this phase, Alice and Bob agree on a common ECDSA public key Q which is used to verify the cooperatively created signatures. Therefore, Alice and Bob generate two private key shares $d_A, d_B \in \mathbb{Z}_n^{\times}$ pseudorandomly. Afterwards, they exchange the corresponding public keys $Q_A = d_A G$ and $Q_B = d_B G$. Both sides now compute the common public key as $Q = d_A Q_B = d_A d_B G$ and $Q = d_B Q_A = d_B d_A G$, respectively. Essentially, they perform a Diffie–Hellman key exchange and thus hold the same public key Q in the end. The full protocol is given in Fig. 1.
   As part of the protocol, Alice and Bob also generate key pairs $(\mathrm{sk}_A, \mathrm{pk}_A)$ and $(\mathrm{sk}_B, \mathrm{pk}_B)$, respectively, for the Paillier cryptosystem and exchange the public keys. Furthermore, they generate and exchange public parameters $(\mathfrak{g}_A, \mathfrak{h}_A, \mathfrak{N}_A)$ and $(\mathfrak{g}_B, \mathfrak{h}_B, \mathfrak{N}_B)$, respectively, for the Fujisaki and Okamoto [10] integer commitment scheme. We can define the fictitious private key $d = d_A d_B$ which is the private key corresponding to the public key Q. Note that none of the two parties ever hold the full private key d nor are they able to compute it.

2. In the second phase, a shared ephemeral secret $k = k_A k_B \in \mathbb{Z}_n^\times$ is generated together with the corresponding public key $R = kG \in E$. Alice and Bob also compute the public keys corresponding to their shares of the ephemeral secret as $R_A = k_A G$ and $R_B = k_B G \in E$. Furthermore, Alice commits to the two values k_A^{-1} and $k_A^{-1} d_A$ in $\mathbb{Z}_n^\times$ by sending the corresponding encryptions under pk_A to Bob.
3. In the final phase, Bob uses the two commitments together with the homomorphic property of the encryption scheme to finally compute the second part of the ECDSA signature s, see Fig. 2.

Note that both the secret signature key d and the ephemeral key k are constructed from two multiplicative shares independently by the two devices. Even if an attacker has full control over one of the devices and can learn or even choose one of the shares, this still does not allow the attacker to recover the full key, since the other device still chooses its share uniformly at random. Thus, the resulting product is indistinguishable from random and therefore unpredictable for the attacker.

3.1 Building blocks

For the protocol to work, it is necessary to prove several facts to the other party using non-interactive zero-knowledge proofs, which we will denote by zkp, see Blum, Feldman, and Micali [4]. Also, the protocol is based on the Paillier cryptosystem, which is an (additively) homomorphic asymmetric cipher described in Paillier [25]. It works as follows:

1. Generation of the public key pk and the private key sk:
   (a) Choose two large primes $\mathbf{p}$ and $\mathbf{q}$ uniformly at random. Set $\mathbf{N} = \mathbf{pq}$ and $\lambda = \operatorname{lcm}(\mathbf{p}-1, \mathbf{q}-1)$.
   (b) Select $\mathbf{g} \in \mathbb{Z}_{\mathbf{N}^2}$, s.t. $\gcd\left(\mathrm{L}\left(\mathbf{g}^\lambda \bmod \mathbf{N}^2\right), \mathbf{N}\right) = 1$ and $\mathrm{L}(u) = u - 1/\mathbf{N}$. The $\mathbf{g}$ does not need to be selected at random for security and Damgård and Jurik [9] suggest to always use $\mathbf{g} = \mathbf{N} + 1$.

   The public key is $\mathrm{pk} = (\mathbf{N}, \mathbf{g})$ and the private key is $\mathrm{sk} = (\lambda)$ or equivalently $(\mathbf{p}, \mathbf{q})$.
2. Encryption of a plain text $m \in \mathcal{M}_{\mathrm{pk}}$ with the message space $\mathcal{M}_{\mathrm{pk}} = \mathbb{Z}_{\mathbf{N}}$:
   (a) Select $r \in \mathbb{Z}_{\mathbf{N}}^*$ uniformly at random.
   (b) Compute the cipher text $c = \mathbf{g}^m \cdot r^{\mathbf{N}} \bmod \mathbf{N}^2$.
3. Decryption of a cipher text $c \in \mathcal{C}_{\mathrm{pk}}$ with the cipher text space $\mathcal{C}_{\mathrm{pk}} \subset \mathbb{Z}_{\mathbf{N}^2}^*$:
   (a) Compute the decrypted plain text

$$m = \frac{\mathrm{L}\left(c^\lambda \bmod \mathbf{N}^2\right)}{\mathrm{L}\left(\mathbf{g}^\lambda \bmod \mathbf{N}^2\right)} \bmod \mathbf{N}.$$

The homomorphic property of a cipher gives rise to an operation

$$+_{\mathrm{pk}} : \begin{array}{rcl} \mathcal{C}_{\mathrm{pk}} \times \mathcal{C}_{\mathrm{pk}} & \longrightarrow & \mathcal{C}_{\mathrm{pk}}, \\ (\mathrm{Enc}_{\mathrm{pk}}(m_1), \mathrm{Enc}_{\mathrm{pk}}(m_2)) & \longmapsto & \mathrm{Enc}_{\mathrm{pk}}(m_1 + m_2) \end{array}$$

which can be implemented as

$$+_{\mathrm{pk}} : (c_1, c_2) \longmapsto c_1 \cdot c_2 \bmod \mathbf{N}^2.$$

Note that the Paillier cryptoscheme is randomized, and consequently, the output of $+_{\mathrm{pk}}$ is only one of many valid encryptions of $m_1 + m_2$. Applying the function $+_{\mathrm{pk}}$ repeatedly defines the function

$$\times_{\mathrm{pk}} : \begin{array}{rcl} \mathcal{C}_{\mathrm{pk}} \times \mathbb{N} & \longrightarrow & \mathcal{C}_{\mathrm{pk}}, \\ (\mathrm{Enc}_{\mathrm{pk}}(m_1), m_2) & \longmapsto & \mathrm{Enc}_{\mathrm{pk}}(m_1 \cdot m_2) \end{array}.$$

which in case of the Paillier cryptoscheme can also be implemented directly as

$$\times_{\mathrm{pk}} : (c_1, k) \longmapsto (c_1)^k \bmod \mathbf{N}^2.$$

Another building block is range bounded integer commitments, which are used inside of the zero-knowledge proofs. These allow a prover to commit to a secret $x \in \mathbb{Z}$ and to prove at the same time that x is inside a certain range as described in Fujisaki and Okamoto [10].

More precisely, in such an integer commitment, a prover $\mathcal{P}$ commits a certain secret value x to a verifier $\mathcal{V}$, such that it cannot be changed later, but at the same time does not reveal the value itself to $\mathcal{V}$. There are two basic requirements for any commitment scheme:

1. The commitment must not leak any information about the committed secret.
2. It must be infeasible to find a $x' \neq x$ which produces the same commitment.

Additionally, the Fujisaki and Okamoto scheme allows $\mathcal{P}$ to prove at the same time that the secret value x falls into a certain range. The two-party ECDSA signature protocol uses a modified version which is non-interactive and has been described in Chan, Frankel, and Tsiounis [7], Damgård and Fujisaki [8], Boudot [5]. It works as follows:

1. The verifier $\mathcal{V}$ generates certain public parameters.
   (a) $\mathcal{V}$ chooses two large Sophie Germain primes $\mathfrak{p}', \mathfrak{q}'$ uniformly at random and $\mathfrak{p}' \neq \mathfrak{q}'$. There are prime numbers $\mathfrak{p}, \mathfrak{q}$, s.t. $\mathfrak{p} = 2\mathfrak{p}' + 1$, $\mathfrak{q} = 2\mathfrak{q}' + 1$.
   (b) $\mathcal{V}$ computes the modulus $\mathfrak{N} = \mathfrak{pq}$.
   (c) $\mathcal{V}$ chooses a random $\mathfrak{g} \in \mathbb{Z}_{\mathfrak{N}}^\times$ with multiplicative order $\mathfrak{p}'\mathfrak{q}'$. The unique cyclic subgroup with order $\mathfrak{p}'\mathfrak{q}'$ of $\mathbb{Z}_{\mathfrak{N}}^\times$ is the set of quadratic residues modulo

$\mathfrak{N}$. A generator $\mathfrak{g}$ of it can be easily computed as $\mathfrak{g} \equiv_{\mathfrak{N}} a^2$ with $a \in \mathbb{Z}_{\mathfrak{N}}^{\times}$ chosen uniformly at random and $\gcd(a-1, \mathfrak{N}) = 1$ and $\gcd(a+1, \mathfrak{N}) = 1$. This holds for most a, see Schmidt [27].

(d) $\mathcal{V}$ chooses $\chi \in \mathbb{Z}_{\mathfrak{p}'\mathfrak{q}'}^{*}$ uniformly at random and computes $\mathfrak{h} = \mathfrak{g}^{\chi} \bmod \mathfrak{N}$.

(e) $\mathcal{V}$ proves to $\mathcal{P}$ that $\mathfrak{h} \in \langle \mathfrak{g} \rangle$. This can be achieved by proving that $\mathfrak{h}$ is a quadratic residue modulo $\mathfrak{N}$.

The public parameters are $(\mathfrak{g}, \mathfrak{h}, \mathfrak{N})$. All other values must stay secret.

2. The prover $\mathcal{P}$ commits to a secret $x \in [0, m]$ and at the same time proves that $x \in \left[-m^3, m^3\right]$.

   (a) $\mathcal{P}$ chooses $r \in \mathbb{Z}_{m\mathfrak{N}}$ uniformly at random.

   (b) $\mathcal{P}$ computes the commitment $z_1 = \mathrm{IC}(x, r) = \mathfrak{h}^x \mathfrak{g}^r \bmod \mathfrak{N}$.

   (c) $\mathcal{P}$ chooses $\alpha \in \mathbb{Z}_{m^3}$, $\gamma \in \mathbb{Z}_{m^3\mathfrak{N}}$ uniformly at random.

   (d) $\mathcal{P}$ computes $z_2 = \mathrm{IC}(\alpha, \gamma) = \mathfrak{h}^{\alpha} \mathfrak{g}^{\gamma} \bmod \mathfrak{N}$.

   (e) $\mathcal{P}$ computes $e = \mathrm{h}(z_2) \bmod m$.

   (f) $\mathcal{P}$ computes $s_1 = \alpha + xe$ and $s_2 = \gamma + re$. These computations are performed in $\mathbb{Z}$. If $s_1 \notin \left[em, m^3 - 1\right]$, $\mathcal{P}$ starts over again.

   The prover $\mathcal{P}$ can now send (z_1, e, s_1, s_2) to the verifier $\mathcal{V}$ as a range bounded commitment to the secret value x.

3. The verifier $\mathcal{V}$ checks a range bounded commitment of the form (z_1, c, s_1, s_2) from the prover $\mathcal{P}$.

   (a) $\mathcal{V}$ checks, that $s_1 \in \left[em, m^3 - 1\right]$.

   (b) $\mathcal{V}$ computes $e' = \mathrm{h}\left(\mathfrak{h}^{s_1} \mathfrak{g}^{s_2} z_1^{-e}\right) \bmod m$.

   (c) $\mathcal{V}$ checks, that $e' = e$.

   The verifier $\mathcal{V}$ is now convinced that $x \in \left[-m^3, m^3\right]$.

The randomizer r can intentionally exceed $\mathfrak{p}'\mathfrak{q}'$, which is the order of $\mathfrak{g}$. This ensures that the commitment z is statistically close to uniform in $\langle \mathfrak{g} \rangle$ and consequently that this commitment scheme has the hiding property, see Damgård and Fujisaki [8].

3.2 The protocol

In Fig. 2, the full two-party ECDSA signature protocol adapted from MacKenzie and Reiter [20] is given. The protocol uses two zero-knowledge proofs to ensure correct execution of the protocol. The first proof π_A, constructed by Alice, proves to Bob the existence of values $x, y \in \left[-n^3, n^3\right]$, such that $xR = R_B$, $(y/x)\,G = Q_A$ and

$$\mathrm{Dec}_{\mathrm{sk}_A}(\alpha_A) \equiv_n x,$$
$$\mathrm{Dec}_{\mathrm{sk}_A}(\beta) \equiv_n y.$$

In other words, Alice proves to Bob that she has properly executed the previous steps in the protocol. The second zero-knowledge proof π_B is used on the other side by Bob to prove to Alice that he has also executed the necessary steps in the protocol and that the operations he performed fit to the operations Alice performed. Specifically, he proves that there are values $x, y \in \left[-n^3, n^3\right]$, $z \in \left[-n^7, n^7\right]$, such that

$$\begin{array}{lcl}
\textbf{Alice }(\mathbf{d_A}, \mathbf{Q_A}, \mathbf{sk_A}) & & \textbf{Bob }(\mathbf{d_B}, \mathbf{Q_B}, \mathbf{sk_B}) \\
k_A \xleftarrow{R} \mathbb{Z}_n^{\times} & & \\
z_A \longleftarrow k_A^{-1} & & \\
\alpha_A \longleftarrow \mathrm{Enc}_{\mathrm{pk}_A}(z_A) & & \\
\beta \longleftarrow \mathrm{Enc}_{\mathrm{pk}_A}(d_A z_A) & \xrightarrow{m, \alpha_A, \beta} & \text{check } \alpha_A, \beta \in \mathcal{C}_{\mathrm{pk}_A} \\
 & & k_B \xleftarrow{R} \mathbb{Z}_n^{\times} \\
\text{check } R_B \in \langle G \rangle & \xleftarrow{R_B} & R_B \longleftarrow k_B G \\
R \longleftarrow k_A R_B & & \\
\pi_A \longleftarrow \mathrm{zkp}_A(R_B, R, \alpha_A, \beta) & \xrightarrow{R, \pi_A} & \text{check } R \in \langle G \rangle, \pi_A \\
 & & r \longleftarrow \mathrm{coord_x}(R) \bmod n \\
 & & z_B \longleftarrow k_B^{-1} \\
 & & c \xleftarrow{R} \mathbb{Z}_{n^5} \\
 & & \sigma \longleftarrow \left(\left(\alpha_A \times_{\mathrm{pk}_A} \mathrm{h}(m)\right) \times_{\mathrm{pk}_A} z_B\right) \\
 & & \qquad +_{\mathrm{pk}_A} \left(\left(\beta \times_{\mathrm{pk}_A} r\right) \times_{\mathrm{pk}_A} d_B z_B\right) \\
 & & \qquad +_{\mathrm{pk}_A} \mathrm{Enc}_{\mathrm{pk}_A}(c \cdot n) \\
 & & \alpha_B \longleftarrow \mathrm{Enc}_{\mathrm{pk}_B}(z_B) \\
\text{check } \sigma \in \mathcal{C}_{\mathrm{pk}_A}, \text{ check } \alpha_B \in \mathcal{C}_{\mathrm{pk}_B}, \pi_B & \xleftarrow{\sigma, \alpha_B, \pi_B} & \pi_B \longleftarrow \mathrm{zkp}_B(m, r, R_B, \alpha_A, \alpha_B, \beta, \sigma) \\
s \longleftarrow \mathrm{Dec}_{\mathrm{sk}_A}(\sigma) \bmod n & & \\
r \longleftarrow \mathrm{coord_x}(R) \bmod n & & \\
\text{publish } (r, s) & &
\end{array}$$

Fig. 2 Generating a two-party ECDSA signature using the modified MacKenzie and Reiter [20] protocol

$xR_B = G$, $(y/x)\,G = Q_B$ and

$$\mathrm{Dec}_{\mathrm{sk}_B}(\alpha_B) \equiv_n x,$$

$$\mathrm{Dec}_{\mathrm{sk}_A}(\sigma) = \mathrm{Dec}_{\mathrm{sk}_A}\Big(\big((\alpha_A \times_{\mathrm{pk}_A} \mathrm{h}(m)) \times_{\mathrm{pk}_A} x\big) +_{\mathrm{pk}_A} \big((\beta \times_{\mathrm{pk}_A} r) \times_{\mathrm{pk}_A} y\big)\Big) + zn.$$

It seems counterintuitive that Bob can argue about decryptions of cipher texts which were encrypted with Alice's public key pk_A. One would expect that this requires knowledge of Alice's secret key sk_A. But Bob is arguing about homomorphic operations with the cipher texts, which are deterministic for him, as he also knows the randomization term zn. In the zero-knowledge proof, he can encode the equality of the two decryptions as equality of two related cipher texts, which Bob can prove without any problems.

We do not present the zero-knowledge proofs π_A and π_B as these can be obtained by straightforward translation of the corresponding proofs in MacKenzie and Reiter [20]. The Fujisaki and Okamoto integer commitment scheme appears as part of the zero-knowledge proofs to implement the aforementioned range bounds for x, y, z.

We finish with an illustration of the correctness of the modified two-party signature scheme:

$$\begin{aligned}
s &= \mathrm{Dec}_{\mathrm{sk}_A}(\sigma)\\
&= \mathrm{Dec}_{\mathrm{sk}_A}\Big(\big((\alpha_A \times_{\mathrm{pk}_A} \mathrm{h}(m)) \times_{\mathrm{pk}_A} z_B\big)\\
&\quad +_{\mathrm{pk}_A} \big((\beta \times_{\mathrm{pk}_A} r) \times_{\mathrm{pk}_A} d_B z_B\big)\\
&\quad +_{\mathrm{pk}_A} \mathrm{Enc}_{\mathrm{pk}_A}(c \cdot n)\Big)\\
&= z_A \mathrm{h}(m)\, z_B + d_A z_A r d_B z_B + c \cdot n\\
&= k_A^{-1} k_B^{-1}(\mathrm{h}(m) + rd)\\
&= k^{-1}(\mathrm{h}(m) + rd)
\end{aligned}$$

Thus, the modified two-party MacKenzie and Reiter signature is indeed a valid ECDSA signature under the private key $d = d_A d_B \in \mathbb{Z}_n^\times$ and the shared ephemeral secret $k = k_A k_B \in \mathbb{Z}_n^\times$.

3.3 Attack scenarios and counter measures

In MacKenzie and Reiter [20], the authors provide a proof of soundness for their protocol, but no rationale for their design choices. In this section, we present several attack scenarios which we designed to illustrate the purpose of the different parts of the protocol. It is quite obvious that Alice is in a much stronger position than Bob in this protocol as she controls the decryption of σ, which contains Bob's secrets, and also the publication of the resulting ECDSA signature. Bob on the other hand is in a rather vulnerable position as he must multiply his secret values with some cipher texts over which he has no control. Consequently, Bob needs strong guarantees that the values encrypted in the cipher texts have been constructed as required by the protocol. These guarantees are given to Bob with the zero-knowledge proof π_A.

3.3.1 Zero-knowledge proofs

We now illustrate what happens when the zero-knowledge proof π_A is missing completely. In this case, Alice is free to choose any $\alpha_A, \beta \in \mathcal{M}_{\mathrm{pk}_A}$. Alice can now sign the malicious message m' with the help of Bob who thinks that he is signing the benign message m. Alice proceeds as follows:

1. Alice follows the first step of protocol, but computes α_A differently as $\alpha_A = z_A \mathrm{h}(m)^{-1}\,\mathrm{h}(m') \bmod n$.
2. Alice and Bob follow the protocol, but Bob does not verify the missing proof π_A in the second to last step.
3. In the last step, Alice computes the ECDSA signature (r, s) as described in the protocol.

The signature Alice just computed is a valid signature for the message m' instead of m as one can see by a simple calculation. Alice has thus succeeded in creating a signature for the malicious message m', while Bob is thinking that he signed the benign message m.

For the zero-knowledge proof π_B, one might notice that Bob sends it together with the encrypted signature σ to Alice. Alice then verifies π_B and afterward publishes the decrypted signature (r, s) without any further verification. This is quite counterintuitive as one would expect that Alice just verifies the decrypted signature (r, s) according to the ECDSA standard before publishing it instead of relying on the zero-knowledge proof π_B. Unfortunately, this approach might cause an information leak on Alice's side. As Bob is no longer restricted by the zero-knowledge proof, he can send arbitrary values to Alice who will decrypt them. Alice must then abort the protocol if the decrypted signature is invalid. In the end, Alice might leak a single bit of information based on the fact whether she aborted the protocol or not. With the zero-knowledge proof in place, Bob only learns that Alice noticed that the zero-knowledge proof is not valid and that he already knows.

3.3.2 Randomization of σ

Another interesting element of the protocol is the randomization of σ which is performed by Bob in the second to last step of the protocol. Bob computes σ as in the protocol with a randomizing term $\mathrm{Enc}_{\mathrm{pk}_A}(c \cdot n)$ where $c \in \mathbb{Z}_{n^5}$ is chosen uniformly at random. Alice later computes $s = \mathrm{Dec}_{\mathrm{sk}_A}(\sigma) \bmod n$, but she can also see the unreduced decryption of σ. Without the randomization, the unreduced decryption leaks information about Bob's secret values.

To work properly, the randomization requires the range proofs for the clear texts x, y of α_A, β in the zero-knowledge proof π_A. These range proofs assure that $x, y \in \left[-n^3, n^3\right]$. When these range proofs are missing and the message space of the Paillier cryptoscheme is large enough, which is the case when using a 4096-bit RSA modulus $\mathbf{N}$, a malicious Alice can easily recover Bob's secret key as follows:

1. Alice executes the first step of the protocol, but computes α_A, β as $\alpha_A = z_A + z_A n^6$ and $\beta = d_A z_A + d_A z_A n^{10}$.
2. Alice executes the protocol together with Bob until the last step.
3. In the last step, Alice receives σ and computes

$$\begin{aligned} s' &= \mathrm{Dec}_{\mathrm{sk}_A}(\sigma) \\ &= z_A \mathrm{h}(m) z_B + z_A n^6 \mathrm{h}(m) z_B + d_A z_A r d_B z_B \\ &\quad + d_A z_A n^{10} r d_B z_B + c \cdot n \end{aligned}$$

4. Alice computes

$$s'' = s' \operatorname{div} n^6 = z_A \mathrm{h}(m) z_B + d_A z_A n^4 r d_B z_B$$

5. Alice retrieves Bob's ephemeral secret z_B by computing

$$\begin{aligned} &\left(s'' \bmod n^4\right) z_A^{-1} \mathrm{h}(m)^{-1} \\ &= \left(\left(z_A \mathrm{h}(m) z_B + d_A z_A n^4 r d_B\right) \bmod n^4\right) z_A^{-1} \mathrm{h}(m)^{-1} \\ &= z_A \mathrm{h}(m) z_B z_A^{-1} \mathrm{h}(m)^{-1} \equiv_n z_B \end{aligned}$$

6. Finally, Alice retrieves Bob's secret key share d_B by computing

$$\begin{aligned} &\left(s'' \operatorname{div} n^4\right) (z_A z_B r d_A)^{-1} \\ &= \left(\left(z_A \mathrm{h}(m) z_B + d_A z_A n^4 r d_B\right) \operatorname{div} n^4\right) (z_A z_B r d_A)^{-1} \\ &= d_A z_A r d_B (z_A z_B r d_A)^{-1} \equiv_n d_B \end{aligned}$$

   Alice now knows all the inverted values, as she retrieved z_B in the step before, r is public, and z_A, d_A are her own secrets.

3.3.3 Experimental verification

Both attack scenarios, the one resulting from a missing zero-knowledge proof π_A and the one resulting from a missing randomization of σ, have been verified in experiments with our prototypic implementation of the two-party ECDSA protocol. We discovered that in case of the missing randomization of σ even a small number of protocol runs (less than 10) will leak the magnitude of Bob's secret key share, such that it can be computed quite precisely by a malicious Alice. Both attack scenarios are now part of the prototype's automated test suite.

3.4 Security analysis

In the following, we discuss the security of both the two-party ECDSA protocol adapted from MacKenzie and Reiter [20] and our pairing protocol.

3.4.1 Security of the MacKenzie and Reiter protocol

In MacKenzie and Reiter [20], the authors give a detailed security analysis for their protocol. We will only summarize their results and provide a list of the assumptions under which the protocol is secure.

MacKenzie and Reiter have proven that their two-party DSA protocol is EUF-CMA (existential unforgeability under chosen message attack) secure against an Alice-compromising or a Bob-compromising attacker under certain assumptions. This means that even if an attacker completely compromises one party and gets access to all its private secret, the success probability to forge a signature is still negligible. To achieve this, both parties must independently check that the message to sign is benign. The assumptions for EUF-CMA security of the two-party ECDSA protocol are the following:

– ECDSA is EUF-CMA secure. This assumption is obvious. If ECDSA is not EUF-CMA secure, an attacker can simply use the existential forgery attack for ECDSA directly on the common public key Q and there is no need anymore to attack the two-party ECDSA protocol itself.
– The Paillier cryptosystem is semantically secure. As proven in Paillier [25], this is the case if and only if the decisional composite residuosity assumption (DCRA) holds. In short, DCRA states that no polynomial time distinguisher exists for $\mathbf{N}$-th residues modulo $\mathbf{N}^2$ with $\mathbf{N} = \mathbf{pq}$ and $\mathbf{p}, \mathbf{q}$ prime. Additionally, it is proven that the DCRA does not hold if the RSA assumption does not hold. While this shows some connection between DCRA and RSA assumption, it does not provide us with a security reduction to the RSA assumption.
– The proofs π_A and π_B are sound. It was proven by MacKenzie and Reiter that the soundness of the proofs π_A and π_B can be reduced to the strong RSA assumption in the DSA case. One can justify the application of their results to the ECDSA case. A further soundness proof for the Fujisaki and Okamoto integer commitment scheme is not required. Note that MacKenzie and Reiter reuse the commitment scheme, but the soundness proof is part of the soundness proof of π_A and π_B.
– The proofs π_A and π_B are statistically zero-knowledge. MacKenzie and Reiter have proven this for DSA, and the applicability of their results in the ECDSA case can be

justified. The statistical zero-knowledge of π_A and π_B does not require any further assumptions.
– The initialization has been correctly performed. All the parameters in the protocol must be correctly generated as required. This can be done by executing the pairing protocol in Fig. 1. We discuss the security of the pairing protocol separately.

In conclusion, the two-party ECDSA signature protocol is EUF-CMA secure under the following assumptions: ECDSA is EUF-CMA secure, the strong RSA assumption holds, and the decisional composite residuosity assumption (DCRA) holds. Furthermore, the random oracle model is used when hash functions appear, and consequently, the security proofs only hold in the random oracle model.

3.4.2 Security of the pairing protocol

During the pairing protocol, each party should prove to the other party that their parameters have been created according to the setup instructions. Three different types of public parameters occur during the pairing:

– The points Q_A and Q_B which correspond to the private key shares d_A and d_B, respectively.
– The Paillier public keys pk_A and pk_B which correspond to the secret keys sk_A and sk_B, respectively.
– The parameters $(\mathfrak{g}_A, \mathfrak{h}_A, \mathfrak{N}_A)$ and $(\mathfrak{g}_B, \mathfrak{h}_B, \mathfrak{N}_B)$ for the Fujisaki and Okamoto commitment scheme.

MacKenzie and Reiter provide a draft for a pairing protocol in an appendix of their article, but they mostly focus on proofs of knowledge for the private key shares for DSA and also for the private keys for the Paillier cryptosystem. Proving the knowledge of a private key is usually done in public key infrastructures when applying for a certificate for a certain public key. Otherwise, a user could bind an arbitrary public key to his identity and signatures which verify under this public key would be attributed to this user even though he has not created them. In the context of Bitcoin, the proofs of knowledge are unnecessary as the key pairs are exclusively used to identify the owner of a certain Bitcoin address. If Alice or Bob send a point Q_A or Q_B for which they do not know the private key share, the Bitcoin address will be inaccessible, but no other consequences will occur. The availability problem can only be solved by introducing a third party as any party can also just forget its shares (see Sect. 6.4).

Consequently, we designed our own pairing protocol (see Fig. 1) which does not contain the unnecessary proofs of knowledge but does consider the integer commitment by Fujisaki and Okamoto which does not appear at all in the draft by MacKenzie and Reiter.

In the following, we discuss the different parts of our pairing protocol. Regarding the points Q_A and Q_B, which are the public keys corresponding to the private key shares d_A and d_B, we only need to ensure during the pairing that $Q_A, Q_B \in \langle G \rangle$. As the used elliptic curve `secp256k1` has a cofactor of 1, this can easily be done by just verifying that Q_A and Q_B are on the elliptic curve.

For the Paillier cryptosystem, proofs of the correct generation of the key pairs and also proofs of knowledge of the private keys are not required. Bob's Paillier key pair $(\mathrm{sk}_B, \mathrm{pk}_B)$ is only used for committing one of Bob's secrets when constructing the zero-knowledge proof π_B. Alice's key pair $(\mathrm{sk}_A, \mathrm{pk}_A)$ on the other hand is also used by Bob as he multiplies his own secrets into the cipher texts encrypted with Alice's public key. Bob protects his secrets by randomizing σ which works no matter how broken Alice's key pair is as the randomization is done by multiplying the rest of σ with an exponentiation of a random element from $\mathbb{Z}^*_{\mathbf{N}_A}$ (the homomorphic addition becomes a multiplication). In the end, each party only endangers its own secrets when incorrectly generating its Paillier key pair.

As mentioned before, the Fujisaki and Okamoto integer commitment scheme is much more delicate. Its parameters must indeed be constructed as required in the setup procedure. The RSA moduli $\mathfrak{N}_A$ and $\mathfrak{N}_B$ must especially be formed from two safe primes, $\mathfrak{g}_A$ and $\mathfrak{g}_B$ must be quadratic residues modulo $\mathfrak{N}_A$ and $\mathfrak{N}_B$, respectively, and $\mathfrak{h}_A \in \langle \mathfrak{g}_A \rangle$ and $\mathfrak{h}_B \in \langle \mathfrak{g}_B \rangle$. Otherwise, the integer commitment scheme does not hide the committed secrets and a malicious verifier can mount a quite efficient attack as described by Kunz-Jacques, Martinet, Poupard, and Stern [17]. Unfortunately and mentioned by Kunz-Jacques et al., achieving provable security is really hard as the construction of $\mathfrak{N}_A$ and $\mathfrak{N}_B$ from safe primes must be proven and no zero-knowledge proof with an acceptable execution time exists for this problem. Consequently, we decided to only apply two attack mitigations which are suggested by Kunz-Jacques et al.:

– Check that $\mathfrak{g}_A, \mathfrak{h}_A$ and $\mathfrak{g}_B, \mathfrak{h}_B$ are quadratic residues modulo $\mathfrak{N}_A$ and $\mathfrak{N}_B$, respectively. This implies that $\mathfrak{h}_A \in \langle \mathfrak{g}_A \rangle$ as the RSA modulus $\mathfrak{N}_A$ is formed by the two safe primes $\mathfrak{p}_A$ and $\mathfrak{q}_A$ with $\mathfrak{p}_A = 2\mathfrak{p}'_A + 1$ and $\mathfrak{q}_A = 2\mathfrak{q}'_A + 1$ and $\mathfrak{p}'_A$ and $\mathfrak{q}'_A$ prime. Hence, $\mathfrak{g}_A$ is a generator of order $\mathfrak{p}'_A\mathfrak{q}'_A$ of the subgroup of quadratic residues modulo $\mathfrak{N}_A$. We can prove $\mathfrak{h}_A \in \langle \mathfrak{g}_A \rangle$ by showing that $\mathfrak{h}_A$ is a quadratic residue modulo $\mathfrak{N}_A$. The same applies to $\mathfrak{g}_B, \mathfrak{h}_B$.
– Check that $\mathfrak{N}_A$ and $\mathfrak{N}_B$ are not prime.

Because of these counter measures, the malicious verifier now needs several hundred protocol runs to recover a single bit of the prover's secret and furthermore a different set of parameters for each bit. Consequently, the described attack becomes infeasible in the setting of our prototype as the para-

Table 1 Required parameter sizes for ECDSA as used in Bitcoin. Parameter sizes chosen for the prototype: 2560 bit for $\mathbf{N}_A$ and $\mathbf{N}_B$

	n	$\mathbf{N}_A$	$\mathbf{N}_B$
ANSSI [1]	256	2048	2048
MacKenzie and Reiter [20]	256	>2304	>1536

meters are fixed in the pairing protocol and each protocol run must be triggered manually by the user on both devices.

3.4.3 Parameter choices for Bitcoin

In Table 1, the parameters sizes required for the two-factor Bitcoin wallet are given. The parameter sizes were chosen based on the established recommendations for key sizes. ECDSA with the curve `secp256k1`, as used in the Bitcoin protocol, uses 256-bit keys. This corresponds to 128 bits of security. To achieve 128 bits of security with RSA, a 2048-bit modulus is required according to ANSSI [1]. Note that others are more pessimistic: NIST [2] recommends at least 3072-bit moduli. On the other hand, there is also an implicit lower bound for the moduli sizes by the protocol itself, since some of the above- mentioned arguments only work when the used parameter sizes are large enough. We decided to use 2560-bit RSA moduli for the Paillier cryptosystem (the smallest multiple of 256 above 2304) which is a good compromise between the different recommendations and also offers acceptable performance on the smart phone.

It should be stressed that we are only talking about short-term security. The Paillier cryptosystem is only used to encrypt private keys and ephemeral secrets for the ECDSA signature scheme, which uses 256-bit keys. The security can be easily increased later to the level provided by 256-bit ECDSA by increasing the RSA modulo size beyond 3072 bit and transferring all Bitcoins to new addresses with new ECDSA key pairs, which were not yet used in the two-party ECDSA signature protocol. Increasing the level of security any further is not possible as the used elliptic curve `secp256k1` is fixed in the Bitcoin protocol.

4 Two-factor Bitcoin wallets

As mentioned in Lipovsky [19], a first Bitcoin stealing online banking Trojan has already been discovered in the wild. When Bitcoin is used by a wider public, attackers might come up with more sophisticated attacks inspired by the attacks on European online banking systems. Therefore, it makes sense to analyze such attacks and to consider the existing counter measures when designing a Bitcoin wallet.

In Sancho, Hacquebord, and Link [26], a common attack on online banking is described. First, the user's computer is compromised by a Trojan, which modifies the victim's DNS resolver and installs an additional attacker controlled certification authority on the system. Consequently, the Trojan can now become a man-in-the-middle between the user and the bank. After the user successfully logged in, the attacker displays a warning to trick the user into installing a malicious app on his phone, which finally allows the attacker to intercept incoming session tokens and transaction numbers. It is important to note that the phone is compromised by tricking the user into installing the spyware app and not by exploiting vulnerabilities in the phone's software.

In Fig. 3, the dataflow when signing a transaction is displayed. To complicate such attacks as far as possible, state-of-the-art online banking systems offer both two-factor authentication and verification over a separate channel. In the commonly used SMS TAN system, the user creates a bank transaction on his computer and then needs to enter a TAN to confirm the transaction. The user receives this TAN via SMS from his bank. The SMS does not only contain the TAN but also the transaction details again and the user can verify them. A compromised computer cannot modify the information in the SMS which allows the user to detect any modifications done to the transaction by an online banking Trojan.

With our Bitcoin wallet, we also provide both two-factor authentication and verification over a separate channel to Bitcoin users. We thus offer users a similar level of security for Bitcoin as they currently have in online banking.

As mentioned before, a Bitcoin address is directly derived from an ECDSA public key and anyone having access to the corresponding private key can spend all Bitcoins stored in this address. Therefore, the only secure way to implement two-factor authentication is to share the private key and to create transaction signatures with a two-party signature protocol. Any other solution would require to store the private key at one place. This place then becomes a single point of failure. Several Bitcoin service providers offer SMS TAN or one-time-password two-factor authentication, but in these cases the service provider stores the private key and becomes a single point of failure. Bitcoin service providers are hardly regulated at the moment, and the when considering the bankruptcy of Mt. Gox, it is clear that leaving the security to the service provider is too risky.

4.1 Connecting the two wallets

We currently assume that both, the desktop and the phone wallet, are located in the same, most likely wireless, local area network. Over the IP connection, the two wallets then establish a TLS channel in which they exchange messages using the Apache Avro serialization library. Currently, we only perform a basic TLS handshake in which the desktop wallet acts as the server. The desktop wallet generates the key pair and the certificate for the TLS connection ad hoc

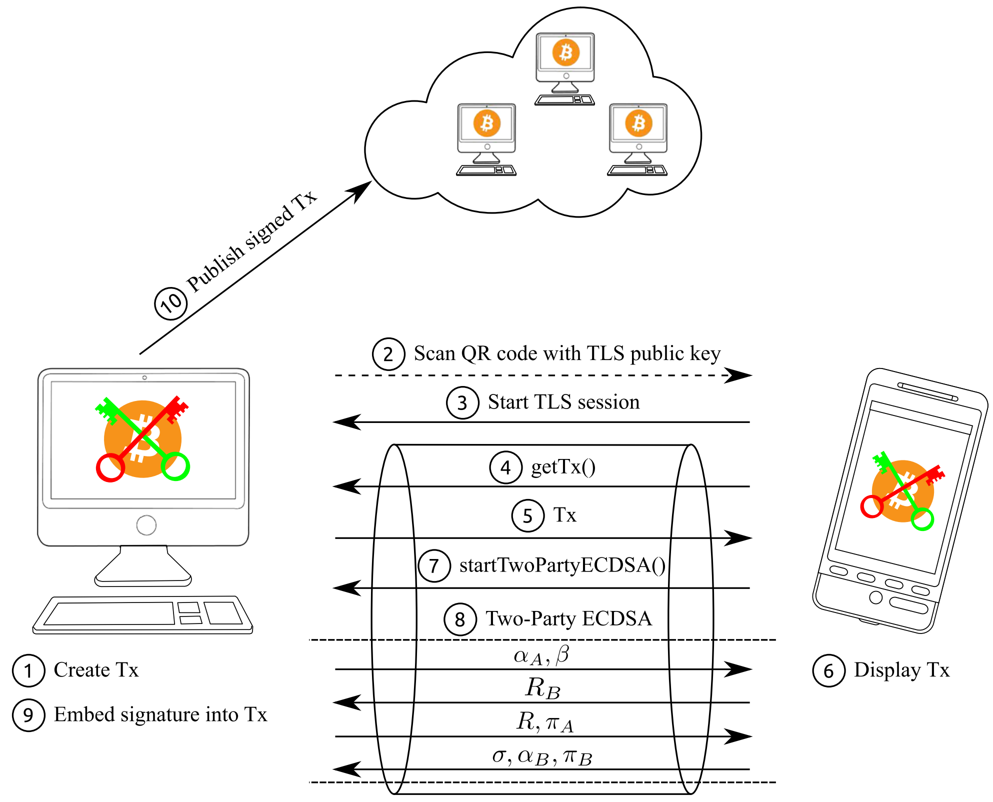


Fig. 3 Signing a Bitcoin transaction with our prototype of a two-factor Bitcoin wallet

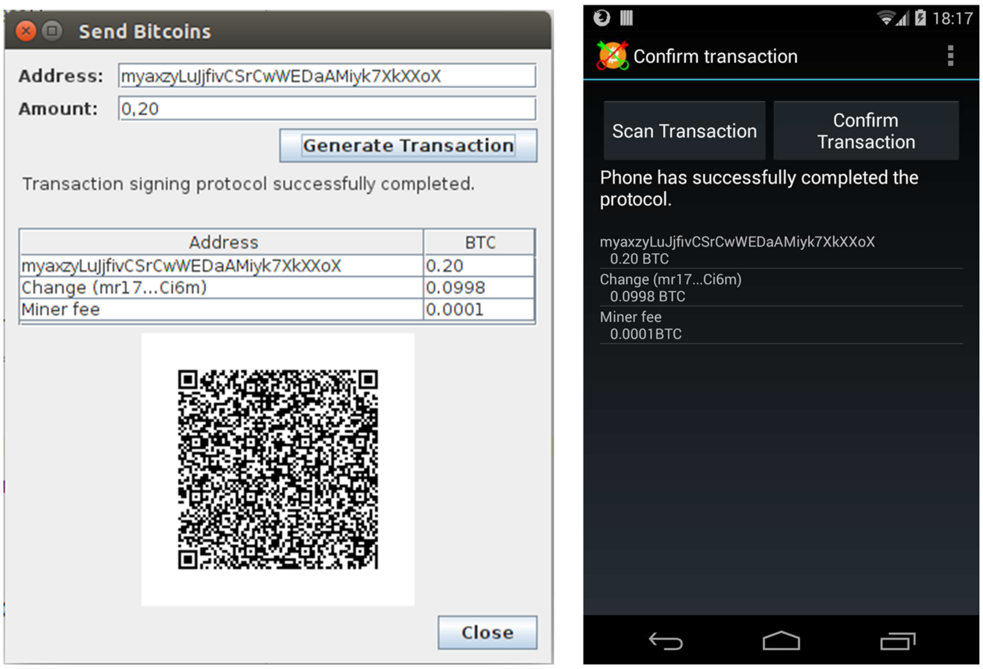
and embeds the public key in a QR code which the phone wallet then scans. With this approach, we prevent man-in-the-middle attacks as the phone receives the desktop's public key via a secure side channel and checks it when establishing the TLS connection. Naturally, one would tend to use a pre-shared key as the QR code provides a secure side channel. Unfortunately, the TLS stacks of both Java and Android do not support TLS with a pre-shared key. To further reduce the attack surface and also to achieve a more rigorous pairing, the two wallets could be connected via Bluetooth. As Bluetooth only allows connections between previously paired devices, network-based attacks would become much harder. Furthermore, the comparison-based pairing of Bluetooth has been proven secure in Lindell [18]. This pairing mode can be used as both devices have screens and the possibility for user input. On the other hand, Bluetooth only pairs the two devices but not directly the two applications. If the Bluetooth connection is used to establish an IP connection, a local attacker might still be able to perform a man-in-the-middle attack. In the end, we decided to not use Bluetooth for our prototype as the configuration is nontrivial and platform specific and more importantly cannot not be driven by the wallet application. The impact on the usability would have been too high.

Before our two-factor authenticated wallet can be used, the desktop wallet and the phone wallet must be paired. During the pairing, the two wallets agree on a Bitcoin address under shared control. The user then needs both devices to spend any Bitcoins from this address. For the pairing, the two wallets execute the protocol depicted in Fig. 1 over a TLS connection which has been established as described before. In this protocol, the two wallets agree on a common ECDSA public key Q. From this public key, both wallets independently derive the user's Bitcoin address and display it to the user. The user must then check that both wallets show the same Bitcoin address to be sure that the pairing was successful and that the Bitcoin address is really under shared control. Without this check, a malicious wallet could trick the user into using a Bitcoin address which it completely controls.

4.2 Creating two-factor authenticated transactions

For our Bitcoin wallet, we use the modified version of the two-party signature protocol by MacKenzie and Reiter [20] as described in Sect. 3. This allows us to share the private key belonging to a Bitcoin address between two different devices and transactions can be signed without ever recombining the private key.

Our two-factor wallet consists of a desktop wallet in form of a Java graphical user interface, and a phone counterpart that is realized as an Android application. Only the desktop



Send Bitcoins

Address: myaxzyLuJjfivCSrCwWEDaAMiyk7XkXXoX

Amount: 0,20

Generate Transaction

Transaction signing protocol successfully completed.

Address	BTC
myaxzyLuJjfivCSrCwWEDaAMiyk7XkXXoX	0.20
Change (mr17...Ci6m)	0.0998
Miner fee	0.0001

Close

18:17

Confirm transaction

Scan Transaction

Confirm Transaction

Phone has successfully completed the protocol.

myaxzyLuJjfivCSrCwWEDaAMiyk7XkXXoX
0.20 BTC
Change (mr17...Ci6m)
0.0998 BTC
Miner fee
0.0001BTC

Fig. 4 The desktop GUI (*left*) and the smart phone GUI (*right*) after completing a transaction

application is a full Bitcoin wallet, which stores and processes all incoming transactions relevant to the user. Consequently, only the desktop wallet can display the transaction history and the current balance. The phone wallet is only required when signing a new transaction. It does not need to connect to the Bitcoin network at all, which makes the implementation much more lightweight. For the full source code, see Mann [21].

When a user wants to send Bitcoins to another person, he starts by creating a Bitcoin transaction with the desktop wallet ①. When the transaction is ready for signing, the desktop wallet displays a QR code which contains the IP address of the desktop wallet and the public key for a TLS connection. The desktop ad hoc generates the key pair and a corresponding server certificate for the TLS connection. Note that the TLS connection has only been added as an additional line of defense and for privacy reasons. The protocol by MacKenzie and Reiter is also secure when the phone and the desktop communicate in clear text.

The user now opens the smart phone wallet and scans the QR code with the phone's camera ②. The smart phone wallet connects to the desktop wallet via the IP address specified in the QR code. The phone wallet establishes a TLS connection with the desktop wallet ③. During the connection setup, the phone wallet verifies that the public key from the desktop's certificate matches the public key in the QR code. This prevents any man-in-the-middle attacks.

Over the secured connection, the phone wallet requests the transaction to sign from the desktop wallet ④ and after receiving it from the desktop ⑤ displays it on the phone's screen ⑥. The user now has the possibility to review the transaction again to make sure that it has not been modified by a compromised desktop wallet.

When the user confirms the transaction on the mobile, the phone wallet asks the desktop wallet to start the two-party signature protocol ⑦. The two wallets then exchange the messages required for the two-party signature protocol over the TLS connection ⑧.

At the end, the desktop wallet holds the correct ECDSA signature for the transaction. It can now embed the signature into the transaction ⑨. Afterward, the desktop wallet publishes the now correctly signed transaction to the Bitcoin network ⑩. Figure 4 shows the desktop and the phone wallet after successfully completing the two-party ECDSA protocol in ⑧.

Note that there is no special protection for the key shares stored on the desktop, the security relies on the design of the MacKenzie and Reiter protocol. An attacker who has control over the desktop (e.g., a banking Trojan) can obtain the desktop's key shares. However, these shares are worthless without the corresponding shares on the phone.

5 Implementation aspects and runtime analysis

In the Bitcoin protocol, the transaction fee (which is payed to the miner) is the difference between the sum of Bitcoins in the transaction inputs and the sum of Bitcoins in the transaction outputs. The inputs actually only reference the outputs of preceding transactions. Consequently, to correctly compute the fee, one needs access to the preceding transactions. In our case, the phone must compute the overpay, which is the fee, itself. Otherwise, the desktop can create a transaction which only contains benign outputs, but spends far too large inputs. The result would be a large fee for the miner and a financial damage for the user.

Implementing full Bitcoin network access is possible as wallet software exists for Android, but would make the phone wallet much more complex. Instead, in our solution, the phone does not only request the transaction to sign from the desktop, but also all transactions that are referenced in the inputs of the transaction to sign. The phone verifies that the hash values of the provided transactions fit the hash values in the transaction inputs. Now the phone can be sure that it has the correct transactions and can use the information from these to compute the overpay in the transaction to sign.

In general, protocols that use zero-knowledge proofs tend to be quite slow. Therefore, we have benchmarked two different prototypes: one prototype using Bitcoin's built-in threshold signature support as described in Sect. 2 and a second one using the two-party signature protocol from Sect. 3. The benchmarks were performed on a core-i5-2520M notebook running Ubuntu 14.04 with OpenJDK, and a Nexus 4 smart phone running Android 4.4.4.

During the benchmark, the execution time of each prototype has been measured for transactions which have one, two, or three inputs. The execution time measured is the time taken by a complete protocol run between the computer and the phone. The results in Table 2 show that the prototype using the two-party signature protocol achieves acceptable runtime, even though Bitcoin's built-in functionality is considerably faster.

On the other hand, when using online banking with SMS TAN, the user has to wait at least several seconds for the SMS. Our execution time is therefore well within the user's expectations.

As mentioned in Sect. 2, Bitcoin's built-in threshold signature support has the disadvantage of increasing the transaction size significantly. We have verified this by recording the size of the resulting transaction during a benchmark. The result in Table 3 shows that the transaction size increases by at least 40 % when using Bitcoin's threshold signatures.

It should be noted that a transaction with only three inputs is already larger than 1000 bytes. Furthermore, larger transactions require a larger transaction fee and have a lower priority to be added to a new block. The priority can be increased by adding an additional fee. Consequently, the solution using Bitcoin's built-in threshold signature support comes with financial costs for the user. In contrast, our solution is transparent to the Bitcoin network and does not influence any fees.

Table 2 Protocol runtime

	1 input	2 inputs	3 inputs
Section 2	0.22s	0.18s	0.25s
Section 3	3.8s	7.4s	11.1s

Table 3 Final size of signed transaction

	1 input	2 inputs	3 inputs
Section 2	370 bytes	696 bytes	1022 bytes
Section 3	257 bytes	438 bytes	619 bytes

6 Future work

As our implementation is only a prototype, there is still some work to do. Besides a thorough code review, we identified the following aspects for future work:

6.1 Execution time

Our prototype already achieves an acceptable execution time when signing a Bitcoin transaction, but there is still some place for improvements. Analyzing the prototype carefully, we found that most of the execution time is used by modular arithmetic on large integers. To reduce it, one could employ more efficient methods for integer multiplication, see, for example, Karatsuba and Ofman [15] or Schönhage and Strassen [28].

6.2 Random numbers

Several versions of Android were shipped with a broken default pseudorandom generator that has not been correctly seeded on start up. This allowed an attacker to recover its state, see Kim, Han, and Lee [16], and lead to Android Bitcoin wallets which generated predictable private keys. In a future version of our wallet, this should be taken into account.

6.3 Integer commitment

The zero-knowledge proofs make use of the integer commitment scheme by Fujisaki and Okamoto [10], which requires a RSA modulus to consist of two safe primes. We have implemented the prime sieve idea from Wiener [30] and achieved a great speedup compared to our first trivial implementation, but on the phone the generation of a safe prime with 2048 bit still takes several minutes. In Damgård and Fujisaki [8], a generalization of the commitment scheme is presented, where the requirement of safe primes has been relaxed to strong primes, which can be generated more easily, see von zur Gathen and Shparlinski [29]. It would be nice to implement this.

6.4 Key derivation, backup and halting attackers

A Bitcoin address is directly derived from the user's ECDSA public key. Thus, an attacker only needs to get control over the corresponding private key to spend all Bitcoins in a certain address. At the same time, when the user loses the private key, all Bitcoins in the corresponding address are lost forever. This is a somewhat special case as in standard public key infrastructures lost signature keys can easily be replaced by creating a new key pair and then issuing a new certificate. The design of Bitcoin poses the special challenge on users to store their private keys securely and at the same time ensuring the availability with the help of backups. In the context of two-factor authentication, the situation is even more involved, since both shares, the desktop and the phone share, need to be stored *separately* in a backup. This is an issue that has to be addressed in the future.

Furthermore, it is desirable to use a new address and consequently a new key pair for each transaction to provide a higher level of privacy to the user. The standard Bitcoin client just generates a new key pair for each transaction and stores it in the user's wallet. Therefore, the wallet file can easily contain hundreds of key pairs. The whole wallet file must now be backed up while being kept secret at the same time.

A solution to this problem is key derivation as described in Wuille [31]. When using such a scheme, all key pairs and consequently all Bitcoin addresses are derived from a single random seed with the help of a secure key derivation function using HMAC-SHA512. Hence, the user only needs to backup the seed securely, which is short enough to be, for example, written down and put into a safe.

A consequence of using our two-factor authentication, which requires two devices to sign a transaction, is that data loss or a halting attack on a single device makes the Bitcoins in the address under shared control inaccessible. Hence, it is highly reasonable to offer support for key derivation by implementing a modified version of the scheme described in Wuille [31]. This would allow a user to easily backup his two-factor authenticated wallet by just securely storing the two seeds used by the two devices, for example with the piece of paper in a safe method.

6.5 Data protection

Currently, both wallets serialize the secrets for the signature protocol and then write them to a file. We have only implemented features related to the protocol, but additional mechanisms like password protection of the wallets and encryption of stored data are not realized, yet.

7 Conclusion

We have shown that one can use the two-party ECDSA signature protocol adapted from MacKenzie and Reiter [20] to realize two-factor authentication for a Bitcoin wallet. Furthermore, we have implemented a prototypic Bitcoin wallet using this protocol. As far as we know, we were able to implement the first fully functional prototype which is compatible with and completely transparent to the Bitcoin production network. Specifically, transactions created by our two-factor wallet are indistinguishable from standard Bitcoin transactions. This transparency is a unique feature that has not been available before and allows users to combine two-factor authentication with CoinJoin, which is a very promising solution for Bitcoin's privacy problem, without experiencing a degraded privacy.

Acknowledgements We would like to thank Michael Nüsken for various useful comments and Mike Hearn for greatly improving the performance of a first version of the prototype by suggesting a bouncy castle version with optimized arithmetic on the curve `secp256k1`. Additionally, we thank the anonymous reviewers for their helpful remarks. This work was funded by the B-IT foundation and the state of North Rhine-Westphalia.

References

1. ANSSI. Mécanismes cryptographiques–Règles et recommandations concernant le choix et le dimensionnement des mécanismes cryptographiques, Rev. 2.03. Agence nationale de la sécurité des systèmes d'information (2014). http://www.ssi.gouv.fr/uploads/2015/01/RGS_v-2-0_B1.pdf
2. Barker, E., Barker, W., Burr, W., Polk, W., Smid, M.: NIST Special Publication 800-57—Recommendation for Key Management-Part 1: General (Revision 3). National Institute of Standards and Technology (2012). http://csrc.nist.gov/publications/nistpubs/800-57/sp800-57_part1_rev3_general.pdf
3. Bitpay Inc. Copay: A secure Bitcoin wallet for friends and companies (2014). www.copay.io
4. Blum, M., Feldman, P., Micali, S.: Proving security against chosen cyphertext attacks. In: Advances in Cryptology: Proceedings of CRYPTO 1988. Santa Barbara, CA, number 403 in Lecture Notes in Computer Science, pp. 256–268. Springer (1988)
5. Boudot, F.: Efficient proofs that a committed number lies in an interval. In: Preneel B (ed) Advances in Cryptology-EUROCRYPT 2000, volume 1807 of Lecture Notes in Computer Science, pp. 431–444. Springer, Berlin, Heidelberg (2000). ISBN 978-3-540-67517-4 (Print) 978-3-540-45539-4 (Online). doi:10.1007/3-540-45539-6_31
6. Certicom Research. Sec 2: Recommended Elliptic Curve Domain Parameters. Technical report, Certicom Corporation (2000)
7. Chan, A., Frankel, Y., Tsiounis, Y.: Easy come-Easy go divisible cash. In: Nyberg K (ed) Advances in Cryptology—EUROCRYPT 98, volume 1403 of Lecture Notes in Computer Science, pp. 561–575. Springer, Berlin, Heidelberg (1998). ISBN 978-3-540-64518-4 (Print) 978-3-540-69795-4 (Online). doi:10.1007/BFb0054154

8. Damgård, I., Fujisaki, E.: A statistically-hiding integer commitment scheme based on groups with Hidden order. In: Zheng Y (eds) Advances in Cryptology—ASIACRYPT 2002, volume 2501 of Lecture Notes in Computer Science, pp. 125–142. Springer, Berlin, Heidelberg (2002). ISBN 978-3-540-00171-3 (Print) 978-3-540-36178-7 (Online). doi:10.1007/3-540-36178-2_8
9. Damgård, I., Jurik, M.: A generalisation, a simplification and some applications of Paillier's probabilistic public-key system. In: Kim K (eds) Public Key Cryptography—PKC 2001, volume 1992 of Lecture Notes in Computer Science, pp. 119–136. Springer, Berlin, Heidelberg (2001). ISBN 978-3-540-41658-6 (Print) 978-3-540-44586-9 (Online). doi:10.1007/3-540-44586-2_9
10. Fujisaki, E., Okamoto, T.: Statistical zero knowledge protocols to prove modular polynomial relations. In: Kaliski J. B. S. (ed) Advances in Cryptology: Proceedings of CRYPTO 1997, Santa Barbara, CA, volume 1294 of Lecture Notes in Computer Science, pp. 16–30. Springer, Berlin, Heidelberg (1997). ISBN 3-540-63384-7. doi:10.1007/BFb0052225
11. Goldfeder, S., Bonneau, J., Felten, E. W., Kroll, J. A., Narayanan, A.: Securing Bitcoin wallets via threshold signatures. preprint, March 2014. http://www.cs.princeton.edu/~stevenag/bitcoin_threshold_signatures.pdf
12. Goldfeder, S., Gennaro, R., Kalodner, H., Bonneau, J., Kroll, J. A. Felten, E. W., Narayanan, A.: Securing Bitcoin wallets via a new DSA/ECDSA threshold signature scheme. preprint, March 2015. http://www.cs.princeton.edu/~stevenag/threshold_sigs.pdf
13. Hearn, M.: Update on mobile 2-factor wallets. Bitcoin Mailing list at http://sourceforge.net (2014). http://sourceforge.net/p/bitcoin/mailman/message/33017648/
14. Ibrahim, M., Ali, I., Ibrahim, I., El-sawi, A.: A robust threshold elliptic curve digital signature providing a new verifiable secret sharing scheme. In: IEEE Computer Society MWCAS03, pp. 276 – 280 Vol. 1. Cairo, Egypt, 27-30 December 2003. ISBN 0-7803-8294-3. doi:10.1109/MWSCAS.2003.1562272
15. Karatsuba, A., Ofman, Y.: Multiplication of multidigit numbers on automata. Soviet Physics–Doklady, **7**(7), 595–596 (1963). translated from Doklady Akademii Nauk SSSR, 145(2), 293–294 July (1962)
16. Kim, S. H., Han, D., Lee, D. H.: Predictability of Android OpenSSL's pseudo random number generator. In: Proceedings of the 2013 ACM SIGSAC conference on Computer and communications security, pp. 659–668. New York, NY, USA (2013). ACM. ISBN 978-1-4503-2477-9. doi:10.1145/2508859.2516706
17. Kunz-Jacques, S., Martinet, G., Poupard, G., Stern J.: Cryptanalysis of an efficient proof of knowledge of discrete logarithm. In: Yung, M., Dodis, Y., Kiayias, A., Malkin, T (eds.) Public Key Cryptography—PKC 2006, volume 3958 of Lecture Notes in Computer Science, pp. 27–43. Springer, Berlin, Heidelberg (2006). ISBN 978-3-540-33851-2 (Print) 978-3-540-33852-9 (Online). doi:10.1007/11745853_3
18. Lindell, Y.: Comparison-based key exchange and the security of the numeric comparison mode in Bluetooth v2.1. In: Fischlin M. (ed.) Topics in Cryptology—CT-RSA 2009, volume 5473 of Lecture Notes in Computer Science, pp. 66–83. Springer, Berlin, Heidelberg (2009). ISBN 978-3-642-00861-0 (Print) 978-3-642-00862-7 (Online). doi:10.1007/978-3-642-00862-7_5
19. Lipovsky, R.: New Hesperbot targets: Germany and Australia (2013). http://www.welivesecurity.com/2013/12/10/new-hesperbot-targets-germany-and-australia/
20. MacKenzie, P., Reiter, M. K.: Two-party generation of DSA signatures. Int. J. Inf. Secur. **2** (3-4), 218–239 (2004). doi:10.1007/s10207-004-0041-0
21. Mann, C.: A prototypic implementation of a two-factor Bitcoin wallet: Source code. GitHub, November 2014. https://github.com/ChristopherMann/2FactorWallet
22. Mann, C., Loebenberger, D.: Two-factor authentication for the Bitcoin protocol. In: Foresti, S. (ed.) Security and Trust Management, volume 9331 of Lecture Notes in Computer Science, pp. 155–171. Springer, Berlin, Heidelberg (2015). ISBN 978-3-319-24857-8 (Print) 978-3-319-24858-5 (Online). doi:10.1007/978-3-319-24858-5_10
23. Nakamoto, S.: Bitcoin: A Peer-to-Peer Electronic Cash System. Cryptography Mailing list at https://metzdowd.com, p. 9 (2008). https://bitcoin.org/bitcoin.pdf
24. NIST. Federal information processing standards publication 180-4—Secure Hash Standard. National Institute of Standards and Technology, March 2012. http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf. Federal Information Processings Standards Publication 180-4
25. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: Stern, J. (ed.) Advances in Cryptology: Proceedings of EUROCRYPT 1999, Prague, Czech Republic, volume 1592 of Lecture Notes in Computer Science, pp. 233–238. Springer, Berlin, Heidelberg (1999). ISBN 3-540-65889-0. doi:10.1007/3-540-48910-X_16
26. Sancho, D., Hacquebord, F., Link, R.: Finding Holes Operation Emmental. Technical report, Trend Micro Incorporated (2014). http://housecall.trendmicro.com/cloud-content/us/pdfs/security-intelligence/white-papers/wp-finding-holes-operation-emmental.pdf
27. Schmidt, J.: Answer to "How to compute a generator of this cyclic quadratic residue group?" (2012). http://math.stackexchange.com/questions/167478
28. Schönhage, A., Strassen, V.: Schnelle Multiplikation großer Zahlen. Computing **7**, 281–292 (1971)
29. von zur Gathen, J., Shparlinski, I.: Generating safe primes. J. Math. Cryptol. **7** (4), 333–365 (2013). ISSN 1862-2984 (Online) 1862-2976 (Print)). doi:10.1515/jmc-2013-5011
30. Wiener, M. J.: Safe Prime Generation with a Combined Sieve. Cryptology ePrint Archive, 2003/186, May 2003. http://eprint.iacr.org/2003/186
31. Wuille, P.: BIP32 Hierarchical deterministic wallets (2014). https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki
32. Ziegeldorf, J. H., Grossmann, F., Henze, M., Inden, N., Wehrle, K.: CoinParty: Secure Multi-Party Mixing of Bitcoins. In Proceedings of the 5th ACM Conference on Data and Application Security and Privacy, CODASPY '15, pp. 75–86. New York, NY, USA, (2015). ACM. ISBN 978-1-4503-3191-3. doi:10.1145/2699026.2699100