TWO-FACTOR AUTHENTICATION FOR THE BITCOIN PROTOCOL

Christopher Mann and Daniel Loebenberger

4 November 2014

Abstract. We show how to realize two-factor authentication for a Bitcoin wallet employing the two-party ECDSA signature protocol adapted from MacKenzie & Reiter (2004). 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.

1. Introduction

Bitcoin (BTC) is a cryptographic currency proposed by Satoshi Nakamoto (2008) in the legendary email 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 *block-chain*, 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in the chain contains the address to which some Bitcoins should be payed, the address from which the Bitcoins should be withdrawn and the amount.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. We describe the details in Section 2. 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 hands on the corresponding wallet, just like in real life. Indeed, Lipovsky (2013) describe 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 build-in functionality for threshold signatures. This has, however, three major disadvantages: First of all, it would be visible in the block-chain 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.

Goldfeder, Bonneau, Felten, Kroll & Narayanan (2014) tried to employ threshold signatures proposed by Ibrahim, Ali, Ibrahim & El-sawi (2003). 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 how to actually realize two-factor authentication for a Bitcoin wallet employing the two-party ECDSA signature protocol adapted from MacKenzie & Reiter (2004). 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.

2. Bitcoin protocol

We will now describe some of the technical details of the Bitcoin protocol as described by Nakamoto (2008). In difference to other e-cash schemes such as the one proposed by Chaum, Fiat & Naor (1990) and many others, Bitcoin was designed to be completely de-central. The Bitcoin network consists of a large number of independent nodes which verify incoming transactions independently of each other. These nodes use a synchronization protocol which is based on a proof-of-work similar to the hashcash system described in Back (2002). With the help of this protocol, the nodes agree on a common transaction history, which is called the Bitcoin *block chain*, see Figure 2.1. A Bitcoin transaction contains the address to which the Bitcoins should be payed, the address from which the Bitcoins should be withdrawn and the amount. Furthermore, the transaction contains a digital signature, which authorizes the transaction, and the public key needed to verify the signature. Bitcoin uses the ECDSA signature scheme, specified by the Accredited Standards Committee X9 (2005) on the elliptic curve `secp256k1` as defined by Certicom Research (2000). All Bitcoin transactions must be correctly signed by the spender. In order to bind Bitcoin addresses and the public keys, the Bitcoin address of a user is directly derived from the user's

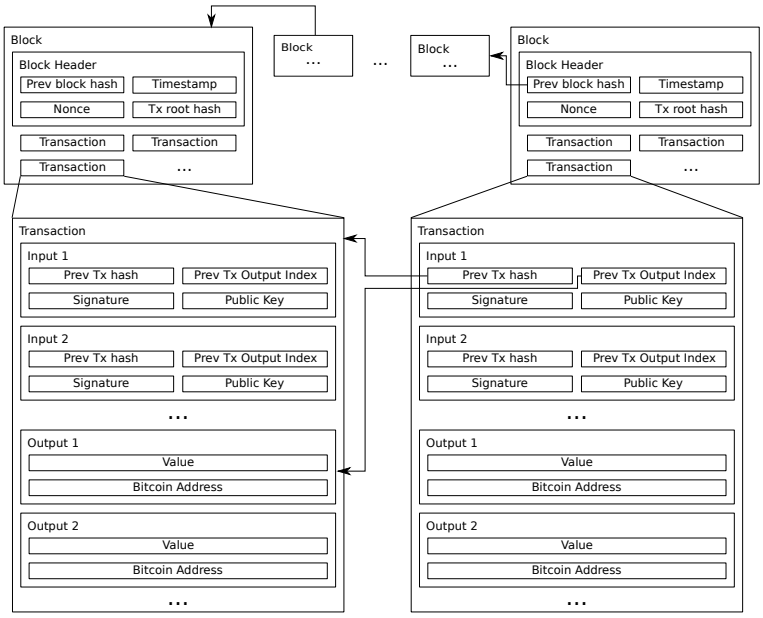

Figure 2.1: Simplified view of the Bitcoin blockchain.

public key by applying a cryptographic hash function to it.

Any Bitcoin transaction actually consists of one or more inputs and outputs. Each output specifies a target address and an amount of Bitcoins to be transferred to this target address. Every input contains the hash of a preceding transaction and an index. Both values together unambiguously identify an output of a preceding transaction. All the Bitcoins from this referenced output are spent by the current transaction. Consequently, every transaction output is only used a single time as an input and is completely spent at this time. This increases the efficiency of the network nodes as these only need to keep track of the unspent outputs instead of all transactions having an impact on the balance of the user's address. Furthermore, any input contains a signature and a public key which must fit the address given in the output referenced by this input. In consequence, if multiple inputs are used, multiple signatures of the transaction must be created, one for each input.

Clearly, the sum of the Bitcoins from all inputs must be greater or equal than the sum of the Bitcoins spent by the outputs. If the sum of the inputs is greater, this is not a problem. Any unused Bitcoins are transferred as a fee to the miner of the block containing this transaction and increase the miner's revenue. Therefore, this will increase the priority of the transaction as the miners will have an incentive to include it into a block.

For ease of exposition, we omitted the fact that Bitcoin uses a scripting language for transactions: In reality, a transaction does not really include a target address or a

signature and a public key, but scripts which contains these as constants. Currently, only a very limited subset of the scripting functionality is actively used in the Bitcoin network and there are plans to restrict the scripting functionality even further to solve the problem of transaction malleability, see Wuille (2014). Currently, transactions are malleable, which means that certain bytes in a transaction can be changed without invalidating the ECDSA signatures. This has been used to attack the Mt Gox Bitcoin exchange. The details are described in Decker & Wattenhofer (2014). Almost all transactions that currently occur, use a standard set of scripts which behave exactly as described above and correspond to the standard use case of transferring Bitcoins from one address to another.

3. Threshold signatures

For a polynomial p, a *$p(t)$-out-of-u threshold signature scheme* allows $p(t)$ members out of a group of u to cooperate in creating a signature for a certain message. At the same time, the scheme is secure against an eavesdropping attacker who compromises less than t parties. A 2-out-of-2 threshold signature scheme is also called a *two-party signature scheme*. In a two-party signature scheme, two parties must work together to create a signature and the scheme is secure against attacks by one of the parties.

For our two-factor Bitcoin wallet, we are interested in a two-party signature scheme which creates signatures that are compatible with ECDSA. The signature algorithm of ECDSA is quite similar to the one of DSA, standardized by NIST (2013). Thus, a DSA-compatible threshold scheme can be ported to ECDSA by replacing the modular operations in DSA by corresponding operations on elliptic curves. Of course, while doing so, the operations in the exponent groups have to be replaced accordingly.

We have searched for threshold signature schemes for both DSA and ECDSA. Several secure and efficient threshold signature schemes exist for modified versions of the ElGamal signature scheme, see for example Harn (1994). Compatibility with DSA or ECDSA on the other hand is harder to achieve, as the the signature algorithm requires the inversion of a secret value and the multiplication of two secret values.

Most threshold signature schemes use polynomial shares similar to Shamir (1979) secret sharing, but the multiplication of polynomial shares does not work well as the multiplication of two polynomials increases the degree of the resulting polynomial. There are several threshold schemes for DSA available, see for example Langford (1995), Gennaro, Jarecki, Krawczyk & Rabin (1996), Wang & Hwang (1997). For ECDSA, Ibrahim *et al.* (2003) presents a $(2t-1)$-out-of-u threshold signature scheme. In Goldfeder *et al.* (2014), this scheme is applied to secure Bitcoin wallets. However, as the authors point out, it is difficult to respect the restrictions on the threshold value in the scheme, rendering it somewhat unsuitable for two-factor authentication. More precisely, it was erroneously assumed that one could further improve the protocol to $(t+1)$-out-of-u by applying the degree reduction protocol from Ben-Or, Goldwasser

& Widgerson (1988) to circumvent the degree doubling caused by the multiplication of two secret sharing polynomials. Unfortunately, the protocol requires $2t + 1 \geq 3$ cooperating parties with secret shares to reduce the polynomial.

In MacKenzie & Reiter (2004), a two-party signature scheme for DSA with a different approach is presented. Instead of working with polynomial shares, the authors use a homomorphic cipher such as the Paillier (1999) cryptosystem. This allows one party to operate with cipher texts of another party's secrets without ever learning about these secrets. In difference to 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.* (2014) came to the same conclusion: In the blog post related to their article they note that the scheme by MacKenzie & Reiter seems to be "close to ideal".

3.1. Two-party ECDSA. We now give a short overview of two-party signatures as described by MacKenzie & Reiter (2004) in the context of ECDSA. For the setup, one fixes a cryptographic hash function h (in our case we use SHA-256, see NIST (2012)) 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, q is a large prime, $a = 0$, $b = 7$ and the cofactor is $h = 1$, see Certicom Research (2000). In order to sign a message $m \in \{0,1\}^*$ in ECDSA, Alice selects pseudorandomly a non-zero 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 $s = k^{-1}(h(m) + rd)$. If $s = 0$, the process is repeated using a new ephemeral key $k \in \mathbb{Z}_n^\times$.

The two-party signature scheme by MacKenzie & Reiter (2004) consists of three different phases for jointly signing a message $m \in \{0,1\}^*$.

Initialization. In this phase an ECDSA key pair (d, Q) is generated. The private key d is multiplicatively shared between the two parties Alice and Bob by selecting $d_A \in \mathbb{Z}_n^\times$ pseudorandomly and computing $d_B = d \cdot d_A^{-1}$ in $\mathbb{Z}_n^\times$. Then $d = d_A d_B$ and Alice gets the share d_A, while Bob gets the share d_B. Both then compute their corresponding public keys $Q_A = d_A G$ and $Q_B = d_B G$. Finally, two key pairs $(\mathrm{sk}_A, \mathrm{pk}_A)$ and $(\mathrm{sk}_B, \mathrm{pk}_B)$ for a homomorphic public key encryption scheme, such as the Paillier (1999) cryptosystem, are generated and distributed to the two parties accordingly.

Constructing an ephemeral key. 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.

Form the signature. 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.

In Figure 3.1, the full two-party ECDSA signature protocol is given. For details on the analysis and the security of this protocol see MacKenzie & Reiter (2004).

For the protocol to be secure, it is necessary to prove to the other side several facts using non-interactive zero-knowledge proofs, see Blum, Feldman & Micali (1988), which we will denote by zkp. Also, there is frequent use of the (additively) homomorphic property of the underlying cipher. For any key pair $(\mathrm{sk}, \mathrm{pk})$, let $\mathcal{M}_{\mathrm{pk}} \subset \mathbb{Z}$ be the message space and $\mathcal{C}_{\mathrm{pk}}$ be the ciphertext space. The homomorphic property of the cipher gives raise to an operation

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

We stress that the encryption function $\mathrm{Enc}_{\mathrm{pk}}$ is randomized such that in the above expression $\mathrm{Enc}_{\mathrm{pk}}(m_1 + m_2)$ denotes *one valid* encryption of the addition of the messages m_1 and m_2. Applying the function $+_{\mathrm{pk}}$ repeatedly defines the function

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

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

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

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

$$\begin{aligned} \mathrm{Dec}_{\mathrm{sk}_B}(\alpha_B) &\equiv_n x, \\ \mathrm{Dec}_{\mathrm{sk}_A}(\sigma) = \mathrm{Dec}_{\mathrm{sk}_A} \Big(&\big((\alpha_A \times_{\mathrm{pk}_A} \mathrm{h}(m)) \times_{\mathrm{pk}_A} x \big) \\ &+_{\mathrm{pk}_A} \big((\beta \times_{\mathrm{pk}_A} r) \times_{\mathrm{pk}_A} y \big) \Big) + zn. \end{aligned}$$

Alice $(\mathbf{d_A}, \mathbf{Q_A}, \mathbf{sk_A})$		**Bob** $(\mathbf{d_B}, \mathbf{Q_B}, \mathbf{sk_B})$
$k_A \xleftarrow{R} \mathbb{Z}_n^\times$		
$z_A \leftarrow k_A^{-1}$		
$\alpha_A \leftarrow \mathrm{Enc}_{\mathrm{pk}_A}(z_A)$		
$\beta \leftarrow \mathrm{Enc}_{\mathrm{pk}_A}(d_A z_A)$	$\xrightarrow{m,\alpha_A,\beta}$	check $\alpha_A, \beta \in \mathcal{C}_{\mathrm{pk}_A}$
		$k_B \xleftarrow{R} \mathbb{Z}_n^\times$
check $R_B \in \langle G \rangle$	$\xleftarrow{R_B}$	$R_B \leftarrow k_B G$
$R \leftarrow k_A R_B$		
$\Pi_A \leftarrow \mathrm{zkp}_A(R_B, R, \alpha_A, \beta)$	$\xrightarrow{R, \Pi_A}$	check $R \in \langle G \rangle, \Pi_A$
		$r \leftarrow \mathrm{coord_x}(R) \bmod n$
		$z_B \leftarrow k_B^{-1}$
		$c \xleftarrow{R} \mathbb{Z}_{n^5}$
		$\sigma \leftarrow \big(\big(\alpha_A \times_{\mathrm{pk}_A} \mathrm{h}(m)\big) \times_{\mathrm{pk}_A} z_B\big)$ $+_{\mathrm{pk}_A} \big(\big(\beta \times_{\mathrm{pk}_A} r\big) \times_{\mathrm{pk}_A} d_B z_B\big)$ $+_{\mathrm{pk}_A} \mathrm{Enc}_{\mathrm{pk}_A}(c \cdot n)$
		$\alpha_B \leftarrow \mathrm{Enc}_{\mathrm{pk}_B}(z_B)$
check $\sigma \in \mathcal{C}_{\mathrm{pk}_A}$, check $\alpha_B \in \mathcal{C}_{\mathrm{pk}_B}$, Π_B	$\xleftarrow{\sigma,\alpha_B,\Pi_B}$	$\Pi_B \leftarrow \mathrm{zkp}_B(m, r, R_B, \alpha_A, \alpha_B, \beta, \sigma)$
$s \leftarrow \mathrm{Dec}_{\mathrm{sk}_A}(\sigma) \bmod n$		
$r \leftarrow \mathrm{coord_x}(R) \bmod n$		
publish (r, s)		

Figure 3.1: Generating a two-party ECDSA signature using the modified MacKenzie & Reiter (2004) protocol.

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

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

Thus, the modified two-party MacKenzie & 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 $k = k_A k_B \in \mathbb{Z}_n^\times$.

3.2. 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. (2014) 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 several disadvantages that are also mentioned by Goldfeder *et al.* (2014): First, it is visible in the public block chain that threshold signatures are used. 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 previous section circumvents these kinds of problems.

4. Two-factor Bitcoin wallets

As mentioned in Lipovsky (2013), 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 & Link (2014), 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.

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.

For our Bitcoin wallet, we use the modified version of the two-party signature protocol by MacKenzie & Reiter (2004) as described in Section 3.1. 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.

4.1. Description of the prototype. 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 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.

In Figure 4.1, the dataflow when signing a transaction is displayed. 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.

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

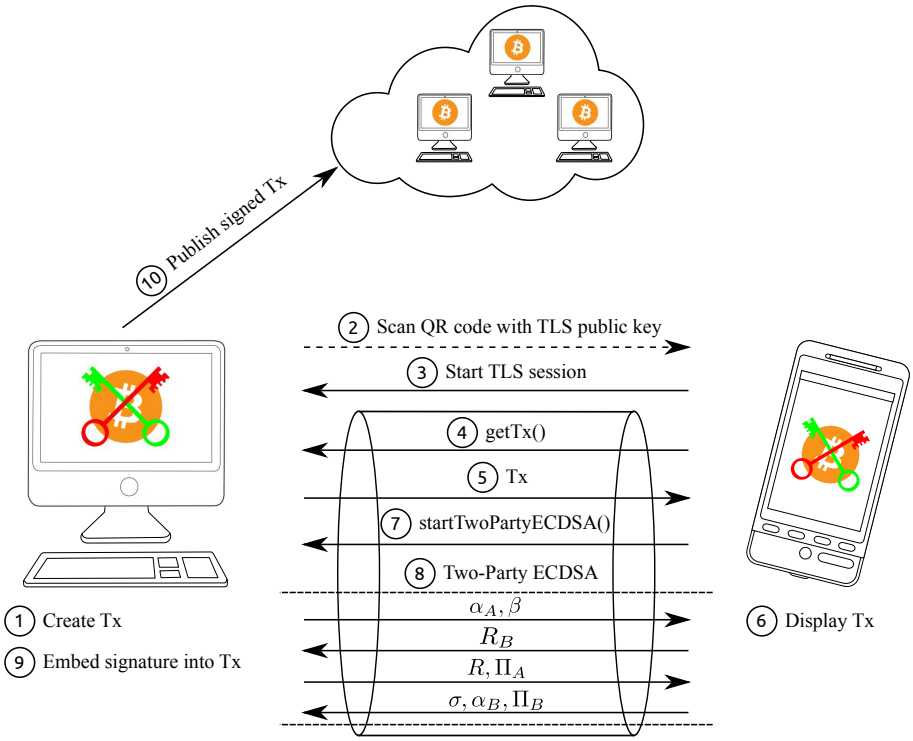

Figure 4.1: The desktop and the smart phone GUI after completing a transaction.

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 is has not been modified by a compromised desktop wallet.

When the user confirms the transaction on the phone, 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. Afterwards, the desktop wallet publishes the now correctly signed transaction to the Bitcoin network ⑩. Figure 4.2 shows the desktop and the phone wallet after successfully completing the two-party ECDSA protocol in ⑧.

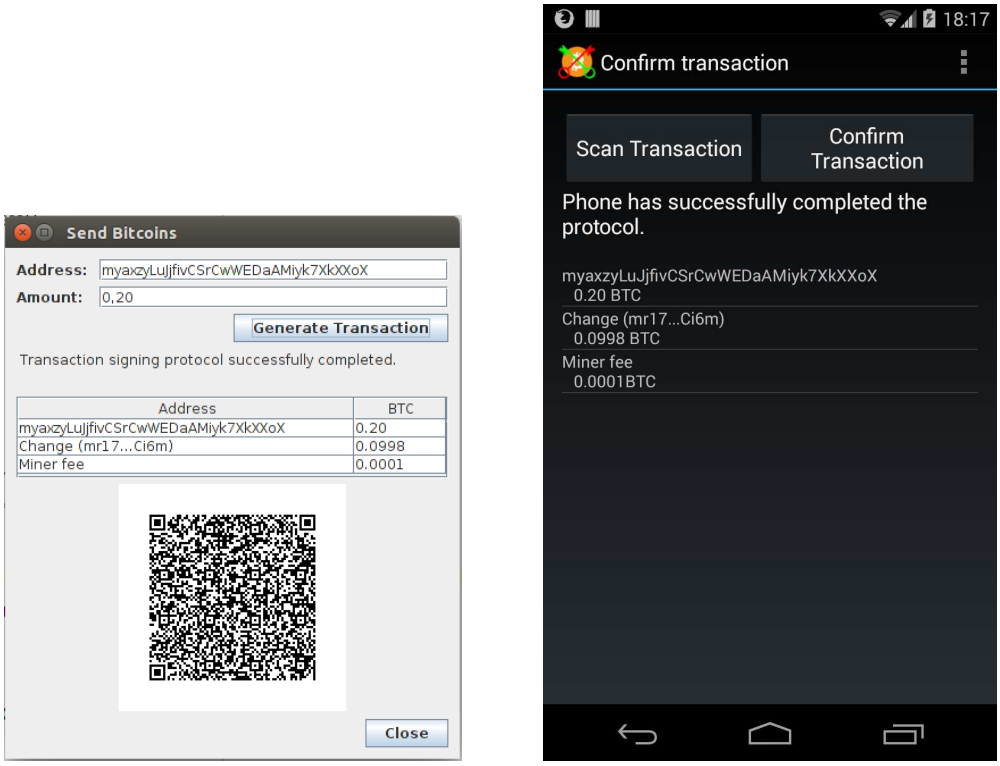

Figure 4.2: The desktop GUI (left) and the smart phone GUI (right) after completing a transaction.

5. Implementation aspects

As explained in Section 2, 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.

5.1. Runtime analysis. In general, protocols that use zero-knowledge proofs tend to be quite slow. Therefore, we have benchmarked two different prototypes: one prototype using the two-party signature protocol from Section 3.1 and a second one using Bitcoin's built-in threshold signature support as described in Section 3.2. 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 Figure 5.1 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.

	1 input	2 inputs	3 inputs
Section 3.1	3.8s	7.4s	11.1s
Section 3.2	0.22s	0.18s	0.25s

Figure 5.1: Protocol runtime

As mentioned in Section 3.2, 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 Figure 5.2 shows that the transaction size increases by at least 40% when using Bitcoin's threshold signatures.

	1 input	2 inputs	3 inputs
Section 3.1	257 bytes	438 bytes	619 bytes
Section 3.2	370 bytes	696 bytes	1022 bytes

Figure 5.2: Final size of signed transaction.

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.

6. Future work

As our implementation is only a prototype, there is still some work to do. Most certainly, before using our software in production, a thorough code review is required to make sure that no implementation mistakes have been made both in the protocol itself and in the supporting code.

6.1. Improving the performance. 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. Especially operations of Paillier crypto system are quite expensive as they require modular arithmetic with 5120 bit integers. Currently, the prototype uses the BigInteger class of the Java platform, which seemingly only contains a straight-forward implementation for integer multiplication. More efficient methods are known for a long time now, see for example Karatsuba & Ofman (1963) or Schönhage & Strassen (1971). We point out that we are dealing with different implementations on the desktop and on the Android smart phone, as Android brings its own BigInteger implementation which utilizes native code.

6.2. Random number generation on Android. Several versions of Android were shipped with a broken default PRNG that has not been correctly seeded on start up. This allowed an attacker to recover the state of the PRNG. The details are described in Kim, Han & Lee (2013). This was fatal for several Android Bitcoin wallets which generated predictable private keys as described in Klyubin (2013). As Android devices often lack security updates, we must expect users with Android versions that are still vulnerable. Consequently, we must use a PRNG provided by our application and we must seed it correctly from a reliable entropy source.

Furthermore, the protocol, especially the ZK proofs, requires a large number of random values. Consequently, a good PRNG with a truly random seed is even more crucial than it would be for ECDSA alone.

6.3. Generation of parameters for the integer commitment scheme. The zero knowledge proofs make use of the integer commitment scheme by Fujisaki & Okamoto (1997), which requires the verifier to generate certain parameters. These parameters include a RSA modulus consisting of two safe primes. The primes must indeed be safe for the scheme to work. The proof Π_A in Figure 3.1, which is verified by the phone is essential for the protocol's security. Therefore, a set of parameters for the integer commitment scheme including the safe primes, which are very expensive to generate, must be generated on the phone. It is possible to generate the parameters only once during the pairing phase in the beginning and reuse them afterwards, but still the generation is very time intensive on the phone. We have implemented the prime sieve idea from Wiener (2003) and we 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 Damgard & Fujisaki (2002), a generalization of the commitment scheme in Fujisaki & Okamoto (1997) is presented, where the the requirement of safe primes has been relaxed to the requirement of strong primes. Generating RSA moduli consisting of strong primes is much cheaper. An efficient method for their generation is presented in von zur Gathen & Shparlinski (2013). Consequently, it would be nice to adapt the protocol to use the generalized scheme by Damgard & Fujisaki (2002).

6.4. Support for multiple addresses. The standard Bitcoin client generates a new address for each payment to provide the user a higher level of privacy. While this approach is not really efficient, it still protects from unsophisticated attackers. Therefore, it is worthwhile to offer support for this. Two general approaches exist. The first one is to generate a new address (and a new key-pair) for each payment. This is done by the standard client. In our case, this means a pairing must be performed each time a payment is received. However, this is currently infeasible. As a workaround, one could generate a set of addresses (and key-pairs) at once during a pairing. This would still require an additional pairing after some time. Alternatively, one could use a key derivation scheme, which allows the desktop and the phone to generate new key-pairs without further interaction from some shared seed. In a future version of the two factor Bitcoin wallet, one of these approaches should definitely be realized.

7. Conclusion

We have shown that one can use the two-party ECDSA signature protocol adapted from MacKenzie & Reiter (2004) to realize two-factor authentication for a Bitcoin wallet. As far as we know, we were able to implement the first fully functional prototype compatible with the Bitcoin production network.

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`. This work was funded by the B-IT foundation and the state of North Rhine-Westphalia.

References

Accredited Standards Committee X9 (2005). ANSI X9.62, Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Standard (ECDSA). Technical report, American National Standards Institute, American Bankers Association.

Adam Back (2002). Hashcash - A Denial of Service Counter-Measure. Technical report. URL `http://www.hashcash.org/papers/hashcash.pdf`.

Michael Ben-Or, Shafi Goldwasser & Avi Widgerson (1988). Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *STOC '88: Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, 1–10. ACM, New York, NY, USA. ISBN 0-89791-264-0. URL http://dx.doi.org/10.1145/62212.62213.

Bitpay Inc. (2014). Copay: A secure Bitcoin wallet for friends and companies. URL www.copay.io.

Manuel Blum, Paul Feldman & Silvio Micali (1988). 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, 256–268. Springer-Verlag. ISSN 0302-9743.

Certicom Research (2000). SEC 2: Recommended Elliptic Curve Domain Parameters. Technical report, Certicom Corporation.

David Chaum, Amos Fiat & Moni Naor (1990). Untraceable Electronic Cash. In *Advances in Cryptology - CRYPTO 88*, Shafi Goldwasser, editor, volume 403 of *Lecture Notes in Computer Science*, 319–327. Springer-Verlag, Berlin, Heidelberg. ISBN ISBN: 978-0-387-97196-4 (Print) 978-0-387-34799-8 (Online). URL http://dx.doi.org/10.1007/0-387-34799-2_25.

Ivan Damgard & Eiichiro Fujisaki (2002). A Statistically-Hiding Integer Commitment Scheme Based on Groups with Hidden Order. In *Advances in Cryptology - ASIACRYPT 2002*, Yuliang Zheng, editor, volume 2501 of *Lecture Notes in Computer Science*, 125–142. Springer-Verlag, Berlin, Heidelberg. ISBN ISBN: 978-3-540-00171-3 (Print) 978-3-540-36178-7 (Online). URL http://dx.doi.org/10.1007/3-540-36178-2_8.

Christian Decker & Roger Wattenhofer (2014). Bitcoin Transaction Malleability and MtGox. *e-print arXive:cs/1403.6676* **abs/1403.6676**, 13. URL http://arxiv.org/abs/1403.66762.

Eiichiro Fujisaki & Tatsuaki Okamoto (1997). Statistical zero knowledge protocols to prove modular polynomial relations. In *Advances in Cryptology: Proceedings of CRYPTO 1997,* Santa Barbara, CA, B. S. Kaliski Jr., editor, volume 1294 of *Lecture Notes in Computer Science*, 16–30. Springer-Verlag, Berlin, Heidelberg. ISBN 3-540-63384-7. ISSN 0302-9743. URL http://dx.doi.org/10.1007/BFb0052225.

Joachim von zur Gathen & Igor Shparlinski (2013). Generating safe primes. *Journal of Mathematical Cryptology* **7**(4), 333–365. ISSN 1862-2984 (Online) 1862-2976 (Print)). URL http://dx.doi.org/10.1515/jmc-2013-5011.

Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk & Tal Rabin (1996). Robust Threshold DSS Signatures. In *Advances in Cryptology - EUROCRYPT 96*, Ueli Maurer, editor, volume 1070 of *Lecture Notes in Computer Science*, 354–371. Springer-Verlag, Berlin, Heidelberg. ISBN ISBN: 978-3-540-61186-8 (Print) 978-3-540-68339-1 (Online). URL http://dx.doi.org/10.1007/3-540-68339-9_31.

Steven Goldfeder, Joseph Bonneau, Edward W. Felten, Joshua A. Kroll & Arvind Narayanan (2014). Securing Bitcoin wallets via threshold signatures. URL http://www.cs.princeton.edu/~stevenag/bitcoin_threshold_signatures.pdf. Preprint.

Lein Harn (1994). Group-oriented (t, n) threshold digital signature scheme and digital multisignature. *Computers and Digital Techniques, IEE Proceedings* **141**(5), 307–313. URL http://dx.doi.org/10.1049/ip-cdt:19941293.

M.H. Ibrahim, I.A. Ali, I.I. Ibrahim & A.H. El-sawi (2003). A robust threshold elliptic curve digital signature providing a new verifiable secret sharing scheme. In *MWCAS03*, 276 – 280 Vol. 1. IEEE Computer Society, Cairo, Egypt. ISBN 0-7803-8294-3. ISSN 1548-3746. URL http://dx.doi.org/10.1109/MWSCAS.2003.1562272.

A. Karatsuba & Yu. Ofman (1963). Multiplication of multidigit numbers on automata. *Soviet Physics–Doklady* **7**(7), 595–596. Translated from Doklady Akademii Nauk SSSR, Vol. 145, No. 2, pp. 293–294, July, 1962.

Soo Hyeon Kim, Daewan Han & Dong Hoon Lee (2013). Predictability of Android OpenSSL's pseudo random number generator. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, 659–668. ACM, New York, NY, USA. ISBN ISBN: 978-1-4503-2477-9. URL http://dx.doi.org/10.1145/2508859.2516706.

Alex Klyubin (2013). Some SecureRandom Thoughts. URL http://android-developers.blogspot.de/2013/08/some-securerandom-thoughts.html.

SusanK. Langford (1995). Threshold DSS Signatures without a Trusted Party. In *Advances in Cryptology - Crypto 95*, D. Coppersmith, editor, volume 963 of *Lecture Notes in Computer Science*, 397–409. Springer-Verlag, Berlin, Heidelberg. ISBN ISBN: 978-3-540-60221-7 (Print) 978-3-540-44750-4 (Online). URL http://dx.doi.org/10.1007/3-540-44750-4_32.

Robert Lipovsky (2013). New Hesperbot targets: Germany and Australia. URL http://www.welivesecurity.com/2013/12/10/new-hesperbot-targets-germany-and-australia/.

Philip MacKenzie & Michael K. Reiter (2004). Two-party generation of DSA signatures. *International Journal of Information Security* **2**(3-4), 218–239. URL http://dx.doi.org/10.1007/s10207-004-0041-0.

Satoshi Nakamoto (2008). Bitcoin: A Peer-to-Peer Electronic Cash System. Cryptography Mailing list at metzdowd.com. URL https://bitcoin.org/bitcoin.pdf. 9 pages.

NIST (2012). *Federal Information Processing Standards Publication 180-4 — Secure Hash Standard*. National Institute of Standards and Technology. URL http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf. Federal Information Processings Standards Publication 180-4.

NIST (2013). FIPS 186-4: Digital Signature Standard (DSS). Technical report, Information Technology Laboratory, National Institute of Standards and Technology.

Pascal Paillier (1999). Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In *Advances in Cryptology: Proceedings of EUROCRYPT 1999,* Prague, Czech Republic, J. Stern, editor, volume 1592 of *Lecture Notes in Computer Science*, 233–238. Springer-Verlag, Berlin, Heidelberg. ISBN 3-540-65889-0. ISSN 0302-9743. URL http://dx.doi.org/10.1007/3-540-48910-X_16.

David Sancho, Feike Hacquebord & Rainer Link (2014). Finding Holes Operation Emmental. Technical report, Trend Micro Incorporated. URL http://housecall.trendmicro.com/cloud-content/us/pdfs/security-intelligence/white-papers/wp-finding-holes-operation-emmental.pdf.

Arnold Schönhage & Volker Strassen (1971). Schnelle Multiplikation großer Zahlen. *Computing* **7**, 281–292.

Adi Shamir (1979). How to Share a Secret. *Communications of the ACM* **22**(11), 612–613.

Chih-Hung Wang & Tzonelih Hwang (1997). (t+1,n) threshold and generalized DSS signatures without a trusted party. In *Proceedings of the 13th Annual Computer Security Applications Conference (ACSAC 97)*, 221–226. IEEE. ISBN ISBN: 0-8186-8274-4. URL http://dx.doi.org/10.1109/CSAC.1997.646193.

Michael J. Wiener (2003). Safe Prime Generation with a Combined Sieve. *Cryptology ePrint Archive* **2003/186**. URL http://eprint.iacr.org/2003/186.

Pieter Wuille (2014). Dealing with malleability. Technical report, Bitcoin Project. URL https://github.com/bitcoin/bips/blob/master/bip-0062.mediawiki.

Christopher Mann
b-it
Universität Bonn
Dahlmannstr. 2
53113 Bonn
Germany
mann@informatik.uni-bonn.de

Daniel Loebenberger
b-it
Universität Bonn
Dahlmannstr. 2
53113 Bonn
Germany
daniel@bit.uni-bonn.de