BlueWallet: The Secure Bitcoin Wallet

Abstract. With the increasing popularity of Bitcoin, a digital decentralized currency and payment system, the number of malicious third parties attempting to steal bitcoins has grown substantially. Attackers have stolen bitcoins worth millions of dollars from victims by using malware to gain access to the private keys stored on the victims' computers or smart phones. In order to protect the Bitcoin private keys, we propose the use of a hardware token for the authorization of transactions. We created a proof-of-concept Bitcoin hardware token: BlueWallet. The device communicates using Bluetooth Low Energy and is able to securely sign Bitcoin transactions. The device can also be used as an electronic wallet in combination with a point of sale and serves as an alternative to cash and credit cards.

Key words: Bitcoin, Transaction, Security, Wallet, Public/Private Key, Authorization

1 Introduction

The digital currency and payment system Bitcoin has become more popular in recent years. As the price of a bitcoin increased to more than 1200 USD in 2013, the number of Bitcoin users and investors increased dramatically.[1] Unlike other payment systems, Bitcoin is not controlled by a central authority. Instead, it is operated by a decentralized authority, the Bitcoin network. This peer-to-peer network collectively handles the creation and transfer of funds using public-key cryptography. As a digital payment system, Bitcoin enables global and secure transactions with low transaction fees.

With its growth in popularity, Bitcoin has also attracted malicious third parties trying to steal other users' bitcoins. In Bitcoin transactions, users receive bitcoins to their Bitcoin *addresses*. To spend the funds associated to a Bitcoin address, control of the corresponding *private key* is needed. Losing access to a private key is equivalent to losing the bitcoins associated to the Bitcoin address. Even though the Bitcoin system itself is protected by strong cryptography, attackers have stolen bitcoins worth millions of dollars by gaining access to the private keys of the victims. The private keys are generally stored on the computers or mobile phones of the users, where they could be exposed to malware and spyware attacks. A study by Litke and Stewart [1] shows that the amount of cryptocurrency-stealing malware has increased with the popularity of Bitcoin.

Whenever the private key is stored on a device connected to the Internet, there is a potential for theft. Our solution is to use a dedicated hardware token to store the private key needed to sign and thus authorize transactions: *BlueWallet*.

[1] We use *Bitcoin* to describe the system and *bitcoin* when we talk about the currency.

This hardware token is used in combination with a device that is connected to the Bitcoin network, like the user's computer. The computer can prepare a Bitcoin transaction, but it cannot sign it. The user can use BlueWallet to review the transaction and sign it. Then, the computer can broadcast the signed transaction to the Bitcoin network. The securely stored private key never leaves the device and is only unlocked if the user correctly enters her PIN.

The hardware token delegates the creation of transactions to another entity and allows independent review of transaction details before signing. It can therefore also be used as an electronic wallet: in combination with a point of sale (POS) connected to the Bitcoin network, the device can be used to directly make Bitcoin payments. BlueWallet offers a mobile and fast solution to securing the user's bitcoins, while at the same time serving as an alternative to cash and credit cards.

2 Bitcoin

Bitcoin is an entirely digital, decentralized currency. The Bitcoin specification was introduced in 2008 by Satoshi Nakamoto [2] and a proof-of-concept client was created in 2009. Bitcoin enables instant global payments. There is no central financial authority like in traditional payment systems. Instead, the whole Bitcoin network acts as the financial authority, using cryptography to control the transfer and creation of money.

2.1 Transactions

In the Bitcoin network, a transaction describes the transfer of a specific amount of bitcoins from one individual to another. Every single Bitcoin transaction is recorded in a public ledger called the *blockchain*. A Bitcoin transaction is a digitally signed data structure that is broadcast in the Bitcoin network [3]. It consists of one ore more *inputs* and one or more *outputs*. Inputs are references to previous transactions and specify the addresses which own the bitcoins that are going to be transferred. Outputs specify the addresses that are going to receive the bitcoins, as well as the amount of bitcoins being transferred.

Each Bitcoin address is associated with a private key that is required to spend the funds assigned to the address. The Bitcoin address is derived from the public key corresponding to the private key. The user signs transactions accessing the funds of the Bitcoin address with her private key and the peers in the network verify the transaction using her public key.

To understand how Bitcoin transactions are signed and verified, it is vital to know how raw bitcoin transactions look like. Figure 1 gives an overview of the transaction structure as defined in the protocol specification:

- **version**: The transaction data format version, a four byte field, with default value 1.
- **tx_in**[ ]: A list of transaction inputs with **tx_in count** elements.
- **tx_out**[ ]: List of the transaction outputs with **tx_out count** elements.

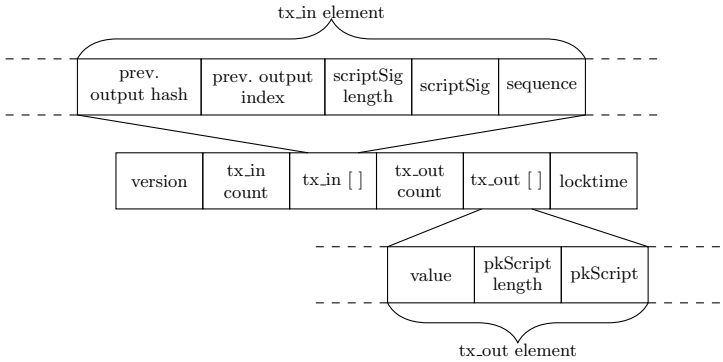

Fig. 1. Transaction packet with different fields.

– **locktime**: The block number or timestamp at which the transaction is locked. By default set to 0, meaning the transaction is immediately locked.

Each *tx_out* output element contains a destination address and the amount of bitcoins that are transferred to this address:

– **value**: Eight byte field holding the transaction value transferred to this output.
– **pkScript**: Script of length **pkScript length** containing the destination address for the bitcoins transferred to this output.

A *tx_in* element comprises a reference to a previous transaction's output and a script containing the signature needed to claim this output:

– **prev. output hash**: 32-byte hash of the previous transaction which is referenced in the input.
– **prev. output index**: Four byte index specifying which output of the referenced previous transaction is used as an input.
– **scriptSig**: Script of length **scriptSig length** containing the signature needed to claim the referenced output and the public key matching the address owning that output.
– **sequence**: Sequence number allowing replacement of transactions.

The *previous output hash* combined with *previous output index* points to a *pkScript* of the referenced previous transaction. This way, the address owning the bitcoins used for the input is determined.

2.2 Bitcoin Cryptography

Bitcoin uses digital signatures to ensure that bitcoins can only be spent by their owner. Ownership of bitcoins is determined by the Bitcoin addresses. The owner of a Bitcoin address holds the private key associated with this address. When creating a transaction to transfer bitcoins from this address, the owner has to

prove that she has the right to do so by providing a signature created with the matching private key. As we have seen, a transaction can have multiple inputs. For a transaction to be valid, the owner must provide a valid signature for each input, thus proving that she has the rights to transfer all of the funds.

The Bitcoin protocol prescribes the use of the Elliptic Curve Digital Signature Algorithm (ECDSA) in order to sign and verify transactions. This class of cryptographic signature algorithms uses algebraic operations on elliptic curves over finite fields. The public key is derived by multiplying the base point of the curve by the private key. The base point of the curve is defined by the curve parameters. Bitcoin uses the secp256k1 curve defined in the standards for efficient cryptography [4]. The security of ECDSA depends on the fact that even though the base point and the public key are public knowledge, it is infeasible to derive the private key from this information.

The primitives provided by ECDSA are *sign()* and *verify()*. The first can be used to calculate a signature S given a message M and the signer's private key p_a, while the latter allows to verify an existing signature given the message and the public key q_a of the signer.

$$S = sign(p_a, M)$$
$$verify(q_a, S, M) = \{true, false\}$$

In Bitcoin, the transaction without any signatures in the inputs is used as the message, hence the signature establishes authenticity and integrity, i.e., the transaction cannot be changed without invalidating the signature attached to the inputs:

$$S = sign(p_a, M) \wedge M' \neq M \rightarrow verify(q_a, S, M') = false$$

In order to create secure ECDSA signatures, a random parameter k is necessary. It is important to select a different k for each signature that is created with the same private key. Otherwise, the private key can be obtained through mathematical backtracking. For example, Sony's implementation of elliptic curve cryptography on their gaming console *Play Station 3* failed to do so, resulting in a compromised private key and full access to the system [5].

3 BlueWallet

The main purpose of BlueWallet is to sign Bitcoin transactions and thus authorize the transfer of bitcoins. In Bitcoin, transactions are usually created by the owner of the transferred bitcoins. It is however possible for another entity to prepare an unsigned transaction that tries to spend these bitcoins. While any entity may create such a transaction, only the owner of the bitcoins can provide the signatures needed to authorize the transaction.

By delegating the preparation of the unsigned transaction to another entity, BlueWallet does not have to be connected to the Bitcoin network. This allows us to build a device with a low power consumption and with small memory

requirements. The user's private key needed to sign a transaction is safely stored on BlueWallet. The private key never leaves BlueWallet and is only unlocked if the user correctly enters her PIN.

There are two applications for BlueWallet. Firstly, it can be used in combination with the user's home computer or smart phone, similar to e-banking solutions. The user can create a Bitcoin transaction on a device which is connected to the Bitcoin network and use BlueWallet to sign it. The private key needed to access the user's funds is no longer stored on her personal computer or smart phone, which is compromised more easily.

The second application is fundamentally different from the first. The transaction is created by an untrusted third party and BlueWallet acts as an electronic wallet. An example of this third party could be the POS in a store, a restaurant or any other place where one would normally pay with cash, debit or credit card. Since the transaction is created by an untrusted party, additional security measures have to be implemented in BlueWallet to minimize the risk incurred by the user. In addition to the signing ability of BlueWallet, the user may review and authorize the transaction independently from the POS and BlueWallet has to ensure that only the authorized bitcoins are transferred.

We will subsequently focus on the second application since it is more challenging. If BlueWallet manages to meet all the necessary requirements for the use with an untrusted POS, it can be used in conjunction with a computer or smart phone owned by the user.

3.1 Creating a Transaction

Assuming a customer in a shop intends to make a payment to the POS using BlueWallet. The POS is connected to the Bitcoin network and will be tasked with the creation of the unsigned payment transaction. In order for the POS to create the transaction, it first needs to learn the customer's address. This is the address whose corresponding private key is stored in the BlueWallet. The POS therefore contacts the BlueWallet and retrieves the address.

Once the POS has the customer's address it scans its local copy of the transaction history for outputs that may be claimed by the address, i.e., earlier transactions that funded the address. The POS will then create a transaction incrementally selecting the found outputs until the desired amount is covered and adding inputs referencing them in the payment transaction. Table 1 shows an example of an unsigned transaction in the same format as it would be transferred from the POS to the BlueWallet. Two outputs are added to the transaction, one transferring the payment amount to the POS, destined to the POS' address (*tx_out[1]*), and the other totaling the remaining bitcoins that are sent back to the address of the BlueWallet (*tx_out[0]*). Should the POS be unable to locate enough outputs to claim the desired amount, it will return an error and abort the transaction creation.

The transaction will be completed by adding the default values for the locktime (0 to lock the transaction immediately) and the sequence in the inputs (`0xffffffff` to disallow replacement). It should be noted that the signature fields scriptSig length and scriptSig are set to an empty string with length 0.

version		01 00 00 00
tx_in count		01
tx_in[0]	prev. output hash	13 cb 3b 56 7d ef 7f fa dc aa 69 de 20 cb 19 09 00 29 02 8b 05 d8 a9 73 d1 5d b5 cf 43 37 a5 a1
	prev. output index	00 00 00 00
	scriptSig length	00
	scriptSig	<empty>
	sequence	ff ff ff ff
tx_out count		02
tx_out[0]	value	c0 2a 99 1c 00 00 00 00
	pkScript length	19
	pkScript	76 a9 14 29 4f db f5 26 0a be 18 48 9b 48 07 f7 ba f0 62 07 70 c3 b7 88 ac
tx_out[1]	value	80 96 98 00 00 00 00 00
	pkScript length	19
	pkScript	76 a9 14 8e e6 7a 65 55 28 b6 1d e2 29 f4 5f c0 16 a0 0f 08 f3 cc 32 88 ac
locktime		00 00 00 00

Table 1. The complete unsigned transaction as prepared by the POS.

Once the unsigned transaction is created, the POS will contact the BlueWallet and transfer the previous transactions as well as the newly created unsigned transaction.

3.2 Unsigned Transaction Verification

Since BlueWallet does not have any connection to the Bitcoin network, it has to take precautions to make sure that the untrusted POS has created a correct transaction.

Once an output has been referenced in a confirmed transaction it is marked as claimed and cannot be claimed again. This means that the entire value associated with an output is always spent in the claiming transaction. This is why bitcoin transactions usually have at least two outputs. One for the address to which a certain amount of bitcoins shall be transferred, and one returning the remaining funds as a new output.

If we have an untrusted party like the POS creating the transaction for us, this could be an issue. If we take a look at a the raw transaction illustrated in Table 1, we observe that the value of an input is not stated in Bitcoin transactions. Therefore, BlueWallet cannot infer the value of the inputs only from the unsigned transaction the POS sent to BlueWallet. Since we do not know how many of our bitcoins are going to be transferred, we would have to rely on the POS to return the correct amount of bitcoins back to our address.

The value of the inputs is usually determined by looking at the outputs of the referenced previous transactions. However, BlueWallet is not connected to the Bitcoin network and cannot look up the previous transactions on its own. Therefore, the POS is required to also send us all of the prior transactions that

are referenced in the inputs of the current transactions. By looking at the outputs of the prior transactions, BlueWallet can determine the sum of all inputs, i.e., how many bitcoins we are going to transfer. BlueWallet compares this sum to the total value of all the outputs. The value of the bitcoins returned to us should be the sum of all our inputs minus the amount of bitcoins transferred to the POS minus an acceptable transaction fee.

BlueWallet, however, has no way to verify that the previous transactions the POS sent were confirmed by the Bitcoin network. How can we make sure the POS did not just change the output values in the prior transactions it sent BlueWallet? To check the correctness of prior transactions, BlueWallet hashes all the received prior transactions. These hashes are compared to the *previous output hashes* in the inputs of the current transaction.

If all the hashes match, we know that the POS sent unmodified prior transactions. Even one single byte-change in a prior transaction would lead to a completely different hash, and thus a rejection of the current transaction.

But what if the POS changed the *previous output hashes* in the current transaction in order to match the hashes of the modified previous transactions? In that case, BlueWallet accepts the current transaction. The Bitcoin network, however, would reject the transaction, since the modified *previous output hashes* do not reference existing prior transactions. In the end, the POS would not receive any bitcoins at all.

This is a strong incentive for the POS to send us the correct prior transactions and to return the correct amount of bitcoins to our address.

3.3 Signing Transactions

Creating the required signatures to authorize a transaction is a rather involved process. To sign a transaction, the owner of the transferred bitcoins has to create a valid signature for every one of the inputs. To create a valid signature for an input, the following steps have to be taken.

First, BlueWallet creates a temporary copy of the unsigned transaction, which is needed to generate the signature. This temporary copy is then modified. The *scriptSig* of the input we want to create the signature for has to be filled with the *pkScript* of the referenced output we want to claim. Remember, the input references a previous transaction's output and this output contains a *pkScript*. This *pkScript* includes the Bitcoin address which owns the output.

Since the output of the previous transaction is owned by the BlueWallet's Bitcoin address, this is a *pkScript* containing our Bitcoin address. The *pkScript* is exactly the same for all outputs owned by our address. In this case, we already encountered it in Table 1. It is included in the transaction change output, since this output will also be owned by our address.

In order to create the signature for the input, BlueWallet replaces *scriptSig* of *tx_in[0]* in Table 1 with the *pkScript* containing our Bitcoin address, and updates *scriptSig length*.

Before BlueWallet can create the signature, it will have to append a so called *hash type* field to the copy of the unsigned transaction. The default value of this four byte field is 1. This *hash type* is called *SIGHASH_ALL* and indicates that

version		01 00 00 00
tx_in count		01
tx_in[0]	prev. output hash	13 cb 3b 56 7d ef 7f fa dc aa 69 de 20 cb 19 09 00 29 02 8b 05 d8 a9 73 d1 5d b5 cf 43 37 a5 a1
	prev. output index	00 00 00 00
	scriptSig length	6b
	scriptSig	48 30 45 02 21 00 b7 28 96 62 40 49 55 c0 87 50 57 2f 8b 6a f4 f4 cf 69 60 c7 67 78 17 64 fd 53 6e f1 99 d0 d8 50 02 20 27 22 90 02 e4 42 a6 1e 8d 98 2a 09 22 57 8f fb 8a cc aa d6 6e 13 d6 e4 80 88 03 6f 71 88 e4 76 01 21 02 84 db aa 32 5f 58 f1 4f 3c 95 c9 55 78 ff 0a 57 10 25 05 eb b8 4c 28 a5 19 f4 f0 e5 07 f8 f4 da
	sequence	ff ff ff ff
tx_out count		02
tx_out[0]	value	c0 2a 99 1c 00 00 00 00
	pkScript length	19
	pkScript	76 a9 14 29 4f db f5 26 0a be 18 48 9b 48 07 f7 ba f0 62 07 70 c3 b7 88 ac
tx_out[1]	value	80 96 98 00 00 00 00 00
	pkScript length	19
	pkScript	76 a9 14 8e e6 7a 65 55 28 b6 1d e2 29 f4 5f c0 16 a0 0f 08 f3 cc 32 88 ac
locktime		00 00 00 00

Table 2. The complete signed transaction created by BlueWallet.

all the outputs are signed. Therefore, each output can only be claimed by its rightful owner. There are also other hash types, but they are not relevant to our use-case.

The modified copy of the unsigned transaction is now double-SHA256 hashed, and the resulting hash is signed with the private key corresponding to our Bitcoin address. The result is a DER-encoded signature [6]. To this signature a one-byte *hash type* has to be added. As the name suggests *scriptSig* is a script that wraps the DER-encoded signature. It starts with a byte indicating the length of the DER-encoded signature including the hashtype-byte. This is followed by the signature and the hashtype-byte itself. Next comes one byte containing the length of the public key. Finally, the public key is added. The final *scriptSig* is shown in the first input in Table 2.

In case the transaction has more than one input, a signature has to be created for every single input. This is done one signature at a time. For every signature a copy of the unsigned transaction is created and only the *scriptSig* of the input we want to create the signature for is temporarily filled with the corresponding *pkScript*. The other inputs are left as is, with empty *scriptSig*.

The valid signed transaction is then created by adding the *scriptSig* generated with the modified copy of the unsigned transaction to the original unsigned transaction shown in Table 1. The empty *scriptSig* field is replaced by the newly generated *scriptSig*, and the *scriptSig length* field is updated accordingly. The complete signed transaction as created by BlueWallet is illustrated in Table 2.

3.4 Verifying Transactions

Once all signatures have been added to the transaction, it is sent back to the POS, which will then verify the transaction's validity.

The POS has to check that its address and the correct value is still listed in the outputs of the signed transaction it received from the BlueWallet. To verify a Bitcoin transaction, its signatures have to be verified. The necessary steps to verify a signature are similar to the steps taken to create a signature. From the signed transaction a transaction equal to the modified copy of the unsigned transaction has to be created. This transaction is then also double-SHA256 hashed, which will lead to the hash that was originally used to create the signature. Using a cryptographic verification algorithm and the public key from the signed transaction, it can now be determined, whether the signature was created from this hash.

If the transaction is not valid, the POS is not going to receive any bitcoins, since the transaction would be rejected by the Bitcoin network. In case the transaction is invalid, the POS aborts the payment process.

If the transaction is found to be valid, the POS releases it into the Bitcoin network. The transaction is verified by other peers and eventually confirmed by the Bitcoin network. A transaction is confirmed when it ends up in the blockchain, the public ledger of the Bitcoin network. This process may take between 10 and 40 minutes. Since waiting this long to complete the payment process is neither in the interest of the POS nor the customer, the POS will have to accept so-called fast payments [7]. Here, the POS does not wait for confirmation by the Bitcoin network. It accepts payments as soon as it sees the transaction being forwarded in the network. Fast payments build upon trusting a transaction to be eventually confirmed by the network. But this might not always be the case.

By accepting fast payments the POS becomes susceptible to double-spend attempts. A double-spend attempt is an attack where the attacker tries to acquire a good or service from a merchant without paying for it. From the POS' view, we could be such an attacker. Our double-spend attempt would include the following steps. Upon receiving the unsigned transaction by the POS, we create a second transaction using the same previous outputs as inputs. The second transaction may transfer the bitcoins to another address, which could be our own. We then sign the original transaction and send it to the POS. If we manage to release the second transaction into the Bitcoin network at the same time, it will be verified by peers in the network and could later be confirmed by the Bitcoin network. Since outputs can only be spent once, the transaction that is supposed to pay the POS will then eventually be rejected by the network. By then, we would have long left the store with the goods.

In order to prevent such double-spend attacks, the POS constantly monitors the Bitcoin network for other transactions spending the outputs chosen for the payment transaction. Furthermore, to secure fast payments the POS could implement the techniques described by [8]. The POS is always connected to a large amount of other peers in the network. For a fast payment transaction to be accepted, a certain percentage of the connected peers must have seen the transaction after a couple of seconds.

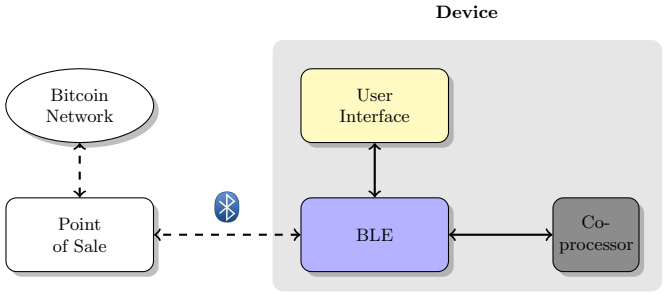

Fig. 2. System overview with contributing components.

4 Implementation

In this section, we illustrate how we created a prototype of BlueWallet with the previously mentioned capabilities.

Figure 2 illustrates the POS scenario and shows the main components of BlueWallet. For communication with the POS BlueWallet incorporates a Bluetooth Low Energy (BLE) module. Compared to classic Bluetooth it provides a considerably reduced power consumption. To process and sign transactions quickly and to improve security, BlueWallet features a co-processor. This co-processor consumes more power than the other parts of the system and therefore immediately enters the stand-by mode whenever it is not used.

4.1 BlueWallet Prototype

The user interface of BlueWallet consists of an OLED display and four buttons. The display is used to show relevant information to the user, the buttons are required for user input. The four buttons are placed next to the four corners of the display. On the right side of the display, we have an *OK* button, used to confirm user input and transactions and a *CANCEL* button, used to cancel user input and reject transactions. On the left side of the display, there is an *UP* and a *DOWN* button, used for selection purposes and for choosing the digits when entering the PIN code.

The Bluetooth Low Energy module is a Bluegiga BLE113 with integrated microcontroller and Bluetooth radio. The BLE113 is the heart of BlueWallet. It is capable of communicating over Bluetooth Low Energy with the POS, interfaces with the microcontroller needed for cryptographic calculations, reacts to user input and controls the display.

The integrated microcontroller in the BLE113 is a 8-bit CC2541 by Texas Instruments. It is a power-optimized chip for BLE applications. The BlueWallet application running on the CC2541 chip is the major building block of our device and implements the state machine, which handles the different states of BlueWallet. It was developed using the BLE software development platform by Texas Instruments.

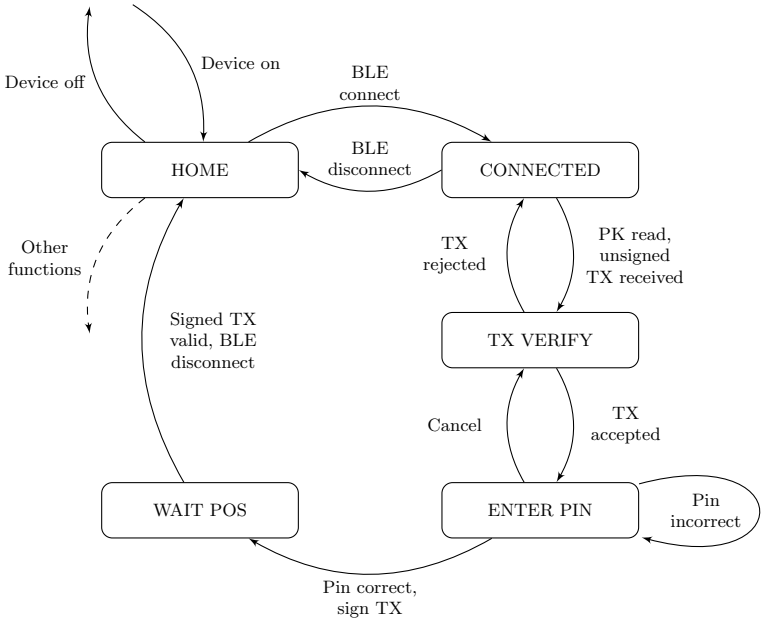

Fig. 3. The important states for the payment process.

For data communication with the POS, BlueWallet uses the Generic Attribute (GATT) layer of the BLE protocol stack. Using a *GATT profile* BlueWallet is able to provide access to GATT *characteristics*. These characteristics contain values that can be written and read by a connected device, depending on the characteristics' properties.

In order to sign transactions, BlueWallet requires more computational resources than the CC2541 could provide. Initial tests with an implementation of ECDSA on this power-saving chip resulted in run-times for a single signature of over 90 seconds, which might be acceptable for non time critical scenarios such as e-Banking, but not for the POS scenario.

Thus, BlueWallet features an STM32F205RE co-processor (STM), a 32-bit microcontroller by STMicroelectronics, that is used for all cryptographic operations. It provides approximately two orders of magnitude speedup, verifying and signing transactions with one input in under a second.

Moreover, by separating the cryptographic domain from the Bluetooth domain, we can improve the overall security of BlueWallet. By only storing the private key needed for signature generation on the STM, we can make sure that even if the Bluetooth connection is compromised the private key is not. Eventually we will use a tamper resistant cryptoprocessor, so that even physical access to BlueWallet would not give access to the encrypted private key. At the time of writing no such processor exists due to the choice of the secp256k1 curve in the Bitcoin protocol.

The BLE113 and the STM communicate over UART via a direct connection. In addition, the BLE113 has the ability to wake up the STM from standby mode. This is necessary because the STM has a significantly higher power consumption when in run mode. The typical supply current when the chip is clocked at 120 MHz lies between 33 and 61 mA. In standby mode the chip only draws around 4 µA. Since BlueWallet is powered by a battery, we generally want to use as little power as possible and thus make use of the standby mode of the STM. The application running on the STM implements the Bitcoin signature scheme described in Section 3.3. It acts like a simple state machine that only reacts to commands and data it receives from the BLE113.

Figure 3 shows the important states for the payment process. Upon starting BlueWallet, the state machine enters the *HOME* state. BlueWallet is now advertising over Bluetooth and the POS is able to establish a Bluetooth Low Energy connection. When a new payment process is started, the POS connects to BlueWallet and the state machine switches to the *CONNECTED* state.

The POS reads the public key of the BlueWallet's owner (characteristic `0xfff1`) and builds the unsigned transaction as specified in Section 3.1. Once the *device state* (`0xfff5`) indicates that BlueWallet is ready to receive data the POS sends the unsigned (`0xfff3`) and the corresponding prior transactions (`0xfff7`). The BLE113 forwards the transactions to the STM which verifies the unsigned and the prior transactions and sends the transaction information back.

The state machine switches to the *TX VERIFY* state and BlueWallet displays the transaction information to the user. In addition to the automatic verification of the unsigned transaction that is done by the STM, the user has to manually confirm the correctness of the transaction. Looking at the display, she can verify the address she is going to transfer bitcoins to, the amount of bitcoins that are transferred and also check the transaction fee.

If the transaction information is correct, the user accepts the transaction and the state machine switches to the *ENTER PIN* state. Each transaction has to be authorized by the owner of BlueWallet with her PIN. This ensures that, even when BlueWallet is lost or stolen, a third party is unable to make payments.

Upon entering the PIN, the BLE113 instructs the STM to sign the transaction. The STM signs the unsigned transaction with the user's private key and returns the signed transaction. The BLE113's state machine switches to the *WAIT POS* state. Now that the signed transaction is ready, which is again signaled by the device state characteristic, the POS reads the signed transaction (`0xfff2`) and uses the user's public key to verify it. If everything is in order, the POS informs BlueWallet that the payment has been accepted by writing the *POS state* (`0xfff6`) characteristic and closes the Bluetooth connection. This causes the state machine to return to the *HOME* state.

For the BlueWallet prototype we created a printed circuit board (PCB) which physically supports and connects all the components. The size of the PCB is restricted by the size of our final device. There are several constraints for the size of the device: the display needs to be large enough to accommodate the necessary information and the buttons should be easily reachable, yet BlueWallet should be small enough to be carried around by the user. The PCB for the prototype has a size of 65 x 30 mm. The buttons and the display are placed on the top

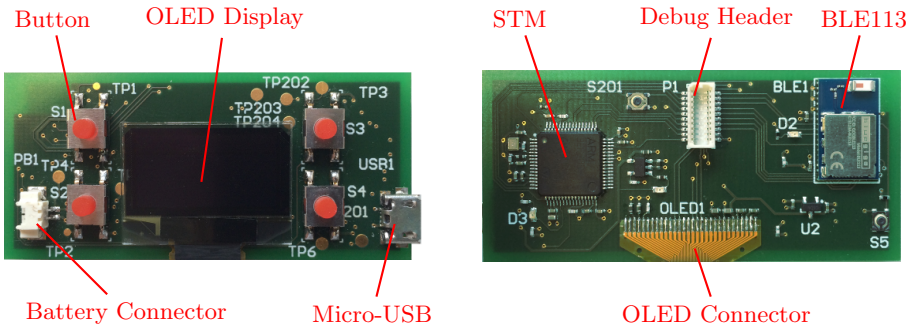

Fig. 4. The top and bottom views of the PCB.

of the PCB, whereas the two microcontrollers and most of the other electrical components are located on the bottom of the PCB.

To determine how much current BlueWallet would draw at most, we summed up the maximum supply currents of the two microcontrollers and the OLED display. The OLED display has a maximum operating current of 28.9 mA, the BLE113 needs 18.2mA and the STM 61mA at most [9–11]. This results in a total maximum supply current needed for BlueWallet of 108.1mA. Therefore, we chose a lithium polymer battery with a capacity of 110mAh to power BlueWallet. This way, BlueWallet will at least run one hour before having to be charged again. A payment process does generally not take longer than 30 seconds. Thus, a user can complete around 100 payment processes before having to charge BlueWallet again. It should be noted that we looked at the maximum operating current of each component. Generally, each component should draw less current resulting in an even longer battery life.

Eventually, the battery will be discharged. To provide the user with a simple way of recharging the battery, we added a micro-USB connector and a battery charger circuit to BlueWallet.

For the prototype, we chose a multi-layer PCB with four layers. It consists of the top and the bottom layer which will hold the electrical components, a power plane, and a ground plane. Components can be connected to these planes using through-hole vias.

The bottom layer of the PCB with the components in place is shown in Figure 4. It should be noted that even though the display is located on the top layer of the PCB, the pads to connect the display are placed on the bottom layer. The display's pins are located on a flexible flat cable. The cable is soldered to the pads on the bottom layer of the PCB and then bent around the edge of the PCB. As a result, the display will come to rest on the top layer of the PCB.

On the top layer illustrated in Figure 4, there are only a few electrical components. Soldered to the top layer are the four buttons, the micro-USB port needed for charging the battery and a 2-pin connector for the battery.

Evaluating the prototype, we found that it takes approximately 1.5 seconds to send the unsigned transaction and one previous transaction to BlueWallet. Upon receiving the transaction details, they are displayed almost instantly by

Fig. 5. The final device including the case for the printed circuit board.

BlueWallet. A thorough review of the transaction details can be done in about 10 seconds. The time it takes to enter the PIN code depends on the length of the PIN code, but generally does not take longer than 10 seconds. The co-processor signs a transaction in less than one second. Then, it again takes around 1.5 seconds to return the signed transaction to the POS. A complete payment process at a POS should therefore take around 20 seconds.

4.2 Point of Sale

To test the BlueWallet in the POS scenario, we implemented the POS on a computer using a CSR 4.0 USB dongle to for Bluetooth Low Energy communication. To establish a connection with the Bitcoin network, our POS uses *bitcoind*, a variant of the reference client. The bitcoind client provides a JSON-RPC API. Our POS application makes use of this RPC interface and is also able to access the Bluetooth Low Energy functions of the USB dongle.

When the POS has established the connection to BlueWallet and read our Bitcoin address, it uses bitcoind to look up the balance of the address and find possible outputs of previous transactions associated to this address that can be used as inputs for the new transaction. With this information it creates the new unsigned transaction. Furthermore, for each input the POS has to send the complete previous transaction to BlueWallet. The POS serializes the previous transactions as illustrated in Figure 6. The total number of previous transactions is followed by the first previous transaction. If there is more than one input in the unsigned transaction, the POS adds the additional previous transactions as well. This data is written to the *prior transaction* characteristic of BlueWallet. Then, the POS sends the unsigned transaction to BlueWallet.

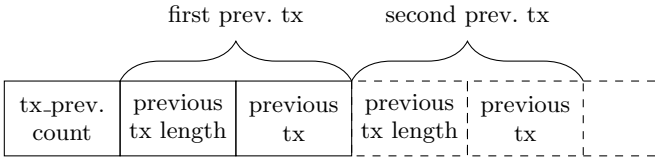

Fig. 6. Previous transactions serialized by the point of sale.

To know when BlueWallet has signed the transaction, the POS subscribes to the *device state* characteristic. The POS will be notified by BlueWallet when the characteristic changes its value. If the value indicates that the transaction has been signed, the POS reads the *signed transaction* characteristic. Then, again using bitcoind, the POS verifies the correctness of the signed transaction and sends it to the Bitcoin network.

5 Related Work

The security of using Bitcoins for fast payment scenarios, like using Bitcoin at a point of sale, where payment confirmation is required immediately, was first analyzed by Karame et al. [7]. The use of Bitcoin at a point of sale using near field communication to exchange payment information with a smart phone was examined by Bronleewe [12]. A proof of concept for a point of sale scenario implementing and expanding upon fast payment security was developed by Bamert et al. [8].

The increase of malware attacks on Bitcoin clients resulting in compromised private keys and theft of bitcoins is discussed by Barber et al. [13]. An analysis of German and US law with regards to theft of bitcoins was conducted by Boehm et al. [14]. They found that traditional criminal law is not well equipped to handle the theft of virtual goods. These findings show that it is vital to protect private keys. Litke and Stewart describe the best practices for storing private keys, including hardware tokens [15]. The benefits of hardware tokens supporting public-key cryptography with regards to e-banking are discussed by Hiltgen [16].

An approach to public-key cryptography called elliptic curve cryptography was proposed independently by Koblitz and Miller [17, 18]. Elliptic curve cryptography is the foundation for the Elliptic Curve Digital Signature Algorithm (ECDSA) which is used by the Bitcoin protocol to sign transactions and is described in detail by Johnson and Menezes [19]. A review of ECDSA in practice to reveal vulnerabilities was done by Bos et al. [20]. They found that repeated per-message signature secrets led to compromised private keys of Bitcoin users.

The benefits of Bluetooth Low Energy when it comes to low energy devices are described by Gomez et al. [21]. Kamath and Lindh measure Bluetooth Low Energy power consumption of a CC2541 chip by Texas Instruments [22].

6 Conclusion

BlueWallet can be used to sign and authorize transactions that are created by the user's computer or smart phone. Using Bluetooth Low Energy to communicate with the entity creating the unsigned transaction we were able to build a device that features a low power consumption and thus is well equipped to be used on the go. Furthermore, by delegating the creation of the unsigned transaction to another entity BlueWallet can directly be used as an electronic wallet in combination with a point of sale. Implementing several security precautions, BlueWallet makes sure that transactions created by an untrusted point of sale can be used to make Bitcoin payments in a store. We found that signing Bitcoin

transactions with BlueWallet is secure and fast. A user can simply pay with Bitcoin by reviewing the transaction information on the screen of BlueWallet and entering her PIN code. Therefore, our electronic wallet is a viable alternative to card based payment methods and cash.

References

1. Litke, P., Stewart, J.: Cryptocurrency-stealing malware landscape (2014) [Online, Retrieved March, 2014].
2. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system. `http://bitcoin.org/bitcoin.pdf` (2008) [Online, Retrieved March, 2014].
3. Decker, C., Wattenhofer, R.: Information propagation in the bitcoin network. In: Peer-to-Peer Computing (P2P). (2013)
4. Standards for Efficient Cryptography (SEC): SEC 2: Recommended elliptic curve domain parameters. Technical report, Certicom Research (2000)
5. Jonathan Fildes: PS3 ECDSA security failure. `http://www.bbc.co.uk/news/technology-12116051` [Online, Retrieved March, 2014].
6. Dubuisson, O.: ASN. 1: communication between heterogeneous systems. Morgan Kaufmann (2001)
7. Karame, G., Androulaki, E., Capkun, S.: Two bitcoins at the price of one? double-spending attacks on fast payments in Bitcoin. IACR Cryptology ePrint Archive (2012)
8. Bamert, T., Decker, C., Elsen, L., Wattenhofer, R., Welten, S.: Have a Snack, Pay with Bitcoins. In: 13th IEEE International Conference on Peer-to-Peer Computing (P2P), Trento, Italy. (2013)
9. Univision Technology Inc.: UG-2864HSWEG01 datasheet, SAS1-9046-B (2009) [Online, Retrieved March, 2014].
10. Bluegiga: BLE113 datasheet v1.2 (2013) [Online, Retrieved March, 2014].
11. STMicroelectronics: STM32F205xx STM32F207xx datasheet, Doc ID 15818 Rev 9 (2012) [Online, Retrieved March, 2014].
12. Bronleewe, D.A.: Bitcoin NFC. Technical report, University of Texas (2011)
13. Barber, S., Boyen, X., Shi, E., Uzun, E.: Bitter to better - How to make bitcoin a better currency. In: Financial Cryptography and Data Security. (2012)
14. Boehm, F., Pesch, P.: Bitcoin: A First Legal Analysis - with Reference to German and American Law. In: Workshop on Bitcoin Research. (2014)
15. Litke, P., Stewart, J.: Enterprise best practices for cryptocurrency adoption (2014) [Online, Retrieved March, 2014].
16. Hiltgen, A., Kramp, T., Weigold, T.: Secure internet banking authentication. Security & Privacy (2006)
17. Koblitz, N.: Elliptic curve cryptosystems. Mathematics of computation (1987)
18. Miller, V.S.: Use of elliptic curves in cryptography. In: Advances in Cryptology-CRYPTO'85. (1986)
19. Johnson, D., Menezes, A., Vanstone, S.: The elliptic curve digital signature algorithm (ECDSA). International Journal of Information Security (2001)
20. Bos, J.W., Halderman, J.A., Heninger, N., Moore, J., Naehrig, M., Wustrow, E.: Elliptic curve cryptography in practice. Microsoft Research. November (2013)
21. Gomez, C., Oller, J., Paradells, J.: Overview and evaluation of Bluetooth Low Energy: An emerging low-power wireless technology. Sensors (2012)
22. Kamath, S., Lindh, J.: Measuring Bluetooth Low Energy power consumption. Texas instruments application note AN092, Dallas (2010)