Simple Schnorr Signature with Pedersen Commitment as Key

Gary Yu

Revised, Feb. 21, 2020

Abstract. In a transaction-output-based blockchain system, where each transaction spends UTXOs (the previously unspent transaction outputs), a user must provide a signature, or more precisely a $scriptSig$ for Bitcoin, to spend an UTXO, which proves the ownership of the spending output. When Pedersen commitment ($g^x h^a$) or ElGamal commitment ($g^x h^a, h^x$) introduced into blockchain as transaction output, for supporting confidential transaction feature, where the input and output amounts in a transaction are hidden, the prior signature schemes such as Schnorr signature scheme and its variants does not directly work here if using the commitment as the public key, since nobody including the committer knows the private key of a $g^x h^a$ when a is not zero, meaning no one knows the c such that $g^c = g^x h^a$. This is a signature scheme which is able to use the $C = g^x h^a$ as the signature public key for any value of a. The signer, proceeding from a random Pedersen commitment $R = g^{k_1} h^{k_2}$, generates a random bit sequence e, by multiplication of a stored private key x with the bit sequence e and by addition of the random number k_1 to get the u, by multiplication of the committed value a with the bit sequence e and by addition of the random number k_2 to get the v, finally constructs $\sigma = (R, u, v)$ as the signature, with the corresponding public key C. In turn, the verifier calculates a Pedersen commitment $S = g^u h^v$, and accepts the signature if $S = RC^e$. For an electronic signature, a hash value e is calculated from a random Pedersen commitment R, the Pedersen commitment C, and from the message m to be signed. This signature scheme will be very helpful in the design of a non-interactive transaction in Mimblewimble.

Keywords: Schnorr signatures, Bitcoin, Mimblewimble, Pedersen commitment, Grin, Gotts

1 Introduction

A signature is a zero-knowledge (abbreviated as ZK in the rest of this paper) proof of knowledge where the prover/s convince the verifier that they know a/some secret information, without revealing the secret info itself. Presently known algorithms for electronic signatures, particularly the Schnorr signature algorithm [Sch90], or the *MuSig* signature scheme [DCC19], is proven secure in the plain public-key model, meaning requires nothing more than that each signer has a certified public key, that is generated individually by each user. The very basic requirement of these signature schemes is that the signer must have the corresponding private key of that public key for a signing.

A commitment scheme is an essential primitive in cryptography, since it allows the committer to put a value in a box, so that nobody has any idea about the actual value (hiding property), but the committer cannot open the commitment in two different ways (binding property). It is also possible to prove relations between committed values, without revealing any additional information about them, expected the fact that they satisfy these relations (zero-knowledge proofs).

For supporting the confidential transaction in some blockchains, such as Monero (getmonero.org), Grin (grin.mw), Beam (beam.mw), Gotts (gotts.tech), and so on, Pedersen commitment scheme [Ped91] is used, where the input and output amounts in a transaction are hidden in commitment. The definition of Pedersen commitment: given the two generators G and H, such that no one knows the discrete log for second generator H with respect to first generator G, meaning no one knows a y such that $y * G = H$, with a random private key x of

the committer, an exemplary commitment scheme to hide the input or output amount a may be defined as:

$$Commitment = x * G + a * H$$

Pedersen commitment scheme is *perfectly hiding* and *computationally binding* [MD17, MRK03]. To prove the security of this *computationally binding* property, given two commitments $C = x * G + a * H$ and $C' = x' * G + a' * H$, such that $C = C'$ and $a \neq a'$, one can calculate $H = \frac{x'-x}{a-a'} * G$, meaning one has computed the discrete logarithm of H. About the *perfectly hiding* (also named as *unconditional hiding*) property, since x is a random element of $\mathbb{Z}_p$, then $x * G + a * H$ is a random element of $\mathbb{G}$, independent of the choice of a.

Alternatively, Switch commitment scheme [RM17] is or can be also used for these blockchains, for example in Grin. Switch commitment scheme constitute a cryptographic middle ground between *computationally binding* and *statistically binding* commitments. Given the third generator J, such that no one knows the discrete log for generator J with respect to first generator G or second generator H, meaning no one knows an y such that $y * G = J$ or $y * H = J$, a Switch commitment scheme may be defined as:

$$Commitment = x' * G + a * H$$

Where,

$$x' = x + Hash(x * G + a * H, x * J) \bmod p$$

Another option is ElGamal commitment scheme, developed by Tahir Elgamal, which is *perfectly binding* and *computational hiding*. An exemplary ElGamal commitment may be defined as $(x * G + a * H, x * H)$ where the left part is a Pedersen commitment.

All these commitment schemes have a common character: the commitment is or include an ellipse curve point, meaning a public key, but no one knows the corresponding private key. Therefore, the commitment cannot be directly used for signature as a key in any presently known algorithms for electronic signatures.

Unlike the UTXO in Bitcoin [Bit08] and other similar cryptocurrencies, which user can provide a direct signature, or more precisely a $scriptSig$ that enables the transaction script to evaluate to true, with specified public key, either with ECDSA presently or with Schnorr signature in the future, to prove the output ownership to spend, the Pedersen commitment output is a public key which nobody (including the committer self) knows its private key, or real public key is hidden in the commitment, so the prior signature schemes cannot work here if directly using the commitment as the public key.

Related Works. Jedusor in Mimblewimble [MW16] found a practical solution to sign with Pedersen commitment as public key in a special condition, he/she realized that a Pedersen commitment to 0 can be viewed as an ECDSA public key, and that for a valid confidential transaction the difference between outputs, inputs and transaction fees must be 0. A sender creating a confidential transaction can therefore sign the transaction with the difference of the outputs and inputs as the public key. An exemplary Mimblewimble transaction scheme [Gri17] with 1 input and 2 outputs may be defined as:

$$(x_i * G + a_i * H) + (excess' + offset * G) = (x_c * G + a_c * H) + (x_r * G + a_r * H) + fee * H$$

Where,

- $(x_i * G + a_i * H)$ is the input commitment owned by sender
- $(x_r * G + a_r * H)$ is the output commitment for receiver
- $(x_c * G + a_c * H)$ is the change commitment for sender
- x_i, x_c, x_r are the private keys; a_i, a_c, a_r are the amounts; fee is the transaction fee
- $offset$ is a random number selected by the sender

The $excess'$ is called as "*public excess*" which is the signature public key and consists of:

$$excess' = (x_c - x_i - offset) * G + x_r * G$$

Where,

- $(x_c - x_i - offset) * G$ is a public key which only sender knows the private key.
- $x_r * G$ is a public key which only receiver knows the private key.

To sign this transaction with $excess'$ as the public key, the Simpler Variants of *MuSig* interactive signature scheme is used, meaning both the sender and the receiver exchanges the public key and public nonce info, then executes a *MuSig* partial signature in both side, then either the sender or the receiver finally aggregate these two partial signatures to get a final joint Schnorr signature, which can be verified exactly as a standard Schnorr signature with respect to a single public key: $excess'$.

Above signature theme for Pedersen commitment input and output in Mimblewimble require a cooperation between sender and receiver, meaning an interactive signing is mandatory for making a transaction, which is slow, complex and hard to use for end users, comparing to those signature schemes which only need one signer.

There were multiple mentions [Lip03, Li12, Pon19] of an extension of Schnorr protocol, which has a similar security property to Schnorr protocol, for a ZK proof of knowledge of a Pedersen commitment opening, meaning to prove one knows both the x and the a of the commitment $x * G + a * H$.

2 The New Signature Scheme

2.1 Description

It is an object of this paper to provide methods for blockchains using Pedersen commitment and for generating signatures that, given essentially the same security guarantees, with any committed value instead of only 0, enable shorter run time due to more simple procedure, in comparison to known cryptographic methods.

The new signature scheme, as illustrated in Figure 1, based on Schnorr signature scheme and denoted as *ComSig* in all following, is parameterized by group parameters $(\mathbb{G}, p, g, h)$ where p is a k-bit integer, $\mathbb{G}$ is a cyclic group of order p, and g is a generator of $\mathbb{G}$, and let h be an element of group $\mathbb{G}$ such that nobody knows $log_g h$, and by a hash function Hash.

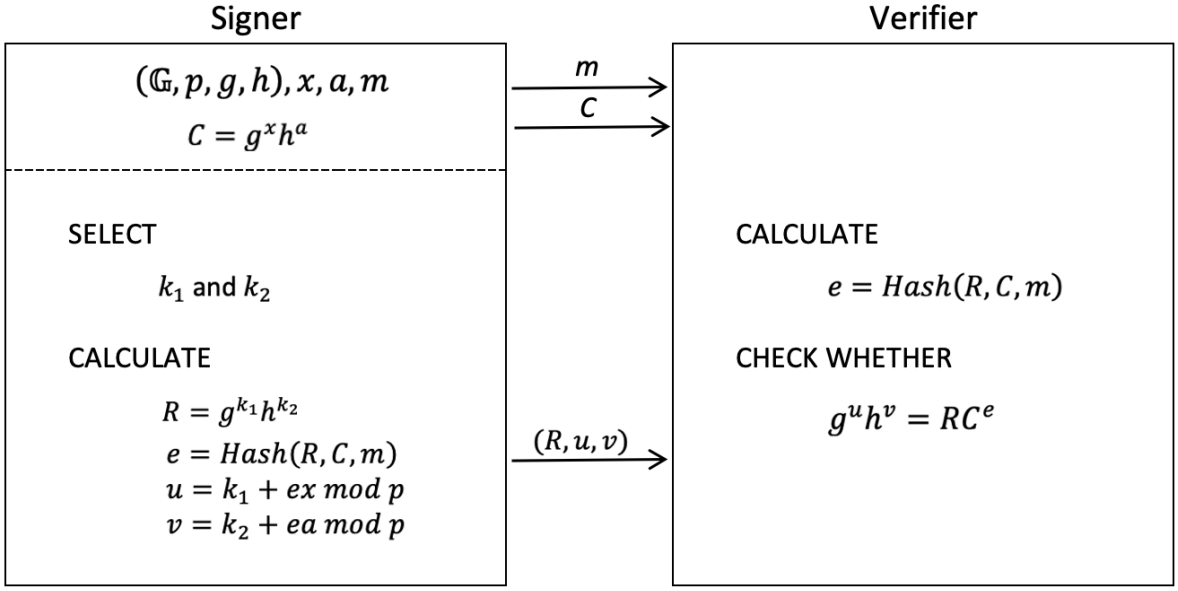

Fig.1. The *ComSig* signature scheme. The signer produces a signature (R, u, v) on message m, the verifier gets it to validate.

Key generation. Signer generates a random private key $x \in \mathbb{Z}_p$ and computes the corresponding public key $X = g^x$.

Pedersen commitment. Signer creates a Pedersen commitment to commit the value a into, computes $C = g^x h^a$.

Signing. Let m be the message to sign. The signer generates two irrelevant random numbers $(k_1, k_2) \in \mathbb{Z}_p$, computes a random Pedersen commitment $R = g^{k_1} h^{k_2}$, and computes

$$e = Hash(R, C, m),$$
$$u = k_1 + ex \bmod p,$$
$$v = k_2 + ea \bmod p,$$

The signature is $\sigma = (R, u, v)$.

Verification. Given a Pedersen commitment C, a message m, and a signature $\sigma = (R, u, v)$, the verifier computes $e = Hash(R, C, m)$, computes Pedersen commitment $S = g^u h^v$, and accepts the signature if $S = RC^e$.

Correctness. Correctness is straightforward to verify, thanks to the *computationally binding* property of Pedersen commitment scheme, meaning the committer cannot open the commitment in two different ways unless one can break the ECDLP, i.e. finding the y such that $g^y = h$.

Because

$$RC^e = (g^{k_1} h^{k_2})(g^x h^a)^e = (g^{k_1} X^e)(h^{k_2} (h^a)^e)$$
$$let\ A = h^a$$

Therefore

$$g^u h^v = RC^e \iff g^u = g^{k_1} X^e \ and\ h^v = h^{k_2} A^e$$

Looking at $g^u = g^{k_1} X^e$, it's exactly the verification for a partial signature (R_1, u) where $R_1 = g^{k_1}$. And same for $h^v = h^{k_2} A^e$, which is exactly the verification for a partial signature (R_2, v) where $R_2 = h^{k_2}$, when taking h as the another generator of $\mathbb{G}$.

Since the verification of $g^u h^v = RC^e$ is equivalent to the parallel verification of $g^u = g^{k_1} X^e \ and\ h^v = h^{k_2} \mathrm{A}^e$ at the same time, this *ComSig* signature is equivalent to two *MuSig* partial signatures. Therefore, this *ComSig* signature scheme has the same security as *MuSig* signature scheme.

Because only the Pedersen commitment C is open as public data in blockchain, both the committer's public key X and A are hidden, here the verifier cannot execute the *MuSig* partial signature verifications.

Note that the *ComSig* signature verification is similar to standard Schnorr signatures (with the Pedersen commitment included in the hash call) with respect to the Pedersen commitment as the public key which nobody knows its private key.

In case of a blockchain transaction is spending multiple UTXOs, the *ComSig* signature scheme is still feasible since all the inputs commitment can be aggregated by $C = \prod_{i=1}^{n} C_i$ where C_i is one input commitment and n is the total number of inputs, and using the aggregated x and a so to keep this aggregated C as the form $g^x h^a$:

$$x = \sum_{i=1}^{n} x_i \bmod p\ ;$$
$$a = \sum_{i=1}^{n} a_i \bmod p\ .$$

where x_i and a_i are from each input commitment $C_i = g^{x_i} h^{a_i}$.

2.2 Attacks against De-randomization

De-randomization Signing. The signer must ensure that the same random value is not reused when signing. Otherwise, anyone can recover this signer's private key.

Assume Alice, holding a key pair (x, g^x), wants to produce a signature. Alice produces two random numbers $(k_1, k_2) \in \mathbb{Z}_p$ but unfortunately $k_2 = k_1$, Alice produces

$$u = k_1 + ex \bmod p,$$
$$v = k_2 + ea \bmod p,$$

and get the signature $\sigma = (R, u, v)$. Then, anyone who gets this signature can now derive

$$x = a + \frac{u-v}{e} \bmod p.$$

To avoid this problem, the signer must ensure that the random value changes unpredictably, or more precisely the generated random values must be irrelevant. Unfortunately, introducing non-repeating randomness requires a secure random number generator at signing time. It can be solved by defining (k_1, k_2) as two splitting parts of one single random number k, therefore it only requires a secure random number generator for one k and that's kept between signing attempts.

3 Multi-Signatures Scheme

3.1 Description

In case a Pedersen commitment is a combination from two Pedersen commitments, and each is committed by different committer, the multi-signatures scheme is needed. Let's use two committers as the example here, for the case of more committers, the method is similar.

Key generation. Alice generates her random private key $x_1 \in \mathbb{Z}_p$ and computes the corresponding public key $X_1 = g^{x_1}$. And Bob also generates his random private key $x_2 \in \mathbb{Z}_p$ and computes the corresponding public key $X_2 = g^{x_2}$.

Pedersen commitment. Alice creates a Pedersen commitment to commit the value a_1 into, computes $C_1 = g^{x_1} h^{a_1}$. Bob also creates a Pedersen commitment to commit the value a_2 into, computes $C_2 = g^{x_2} h^{a_2}$. And they combine their commitment to get a $C = C_1 C_2$ and use C as the transaction output.

Signing. Alice generates two irrelevant random numbers $(k_1, k_2) \in \mathbb{Z}_p$, computes a random Pedersen commitment $R_1 = g^{k_1} h^{k_2}$. Bob generates two irrelevant random numbers $(q_1, q_2) \in \mathbb{Z}_p$, computes a random Pedersen commitment $R_2 = g^{q_1} h^{q_2}$. Alice and Bob exchange their knowledge of R_1, R_2 and X_1, X_2, calculate $R = R_1 R_2$ and $X = X_1 X_2$.
Let m be the message to sign.
Alice computes $e = Hash(R, C, m)$,

$$u_1 = k_1 + ex_1 \bmod p,$$
$$v_1 = k_2 + ea_1 \bmod p.$$

Bob computes $e = Hash(R, C, m)$,

$$u_2 = q_1 + ex_2 \bmod p,$$
$$v_2 = q_2 + ea_2 \bmod p.$$

Alice and Bob exchange their (u_1, v_1) and (u_2, v_2), and calculate the aggregated signature as $\sigma = (R, u, v)$, where $u = u_1 + u_2$ and $v = v_1 + v_2$.

Verification. Given a Pedersen commitment C, a message m, and a signature $\sigma = (R, u, v)$, the verifier computes $e = Hash(R, C, m)$, computes Pedersen commitment $S = g^u h^v$, and accepts the signature if $S = RC^e$.

The final aggregated signature of Alice and Bob is exactly same as a single *ComSig* signature, just looks like someone knows a private key x such that $x = x_1 + x_2$, and also knows a $r_1 = k_1 + q_1$ and a $r_2 = k_2 + q_2$ such that $R = g^{r_1} h^{r_2}$. Therefore, as seen above, the signature verification is exactly same as a single *ComSig* signature verification.

Correctness.

Because

$$\begin{aligned} RC^e &= (R_1 R_2)(C_1 C_2)^e \\ &= (g^{k_1} h^{k_2} g^{q_1} h^{q_2})(g^{x_1} h^{a_1} g^{x_2} h^{a_2})^e \\ &= g^{k_1+q_1}(g^{x_1} g^{x_2})^e h^{k_2+q_2}(h^{a_1+a_2})^e \\ &= g^{k_1+q_1} X^e h^{k_2+q_2}(h^{a_1+a_2})^e. \end{aligned}$$

$$let\ A = h^{a_1+a_2}$$

$$let\ r_1 = k_1 + q_1\ and\ r_2 = k_2 + q_2$$

Therefore

$$g^u h^v = RC^e \Leftrightarrow\ \ g^u = g^{r_1} X^e\ and\ h^v = h^{r_2} A^e$$

Looking at $g^u = g^{r_1} X^e$, it's exactly the verification for a *MuSig* aggregated signature (R', u) where $R' = g^{r_1}$ and $u = u_1 + u_2$. And same for $h^v = h^{r_2} A^e$, which is exactly the verification for an aggregated signature $(R^{"}, v)$ where $R^{"} = h^{r_2}$ and $v = v_1 + v_2$, when taking h as the another generator of $\mathbb{G}$.

Since the verification of $g^u h^v = RC^e$ is equivalent to the parallel verification of $g^u = g^{r_1} X^e\ and\ h^v = h^{r_2} \mathrm{A}^e$ at the same time, the *ComSig* multi-signatures is equivalent to two *MuSig* multi-signatures. Therefore, the *ComSig* multi-signatures scheme has the same security as *MuSig* multi-signatures scheme. Also, those attacks against simpler variants and de-randomization in *MuSig* scheme also applies here.

3.2 Rogue-Key Attack

The attacker can easily give a rogue-key attack on above multi-signature scheme, if he/she use $C_2 = C_2' C_1^{-1}$ so that $C = C_1 C_2 = C_1(C_2' C_1^{-1}) = C_2'$, then the attacker is able to construct a signature just by self if he/she owns the private keys of C_2'.

Similar as *MuSig* multi-signatures scheme, the fix solution to this rogue-key attack may use a weighted aggregated commitment instead of the simple combination, as described below.

- For multiple Pedersen commitment $C_i = g^{x_i} h^{a_i}$ where $i \in [0..n]$, n is the total number.
- Each committer shares his/her C_i so as to calculate the $C = \sum_{i=1}^{n}(f_i * C_i)$, where $f_i = Hash([C_1, C_2, \ldots, C_n], C_i)$.
- Each committer generates and shares one random Pedersen commitment $R_i = g^{k_{1i}} h^{k_{2i}}$, calculates the combination $R = \sum_{i=1}^{n} R_i$, and calculates $e = Hash(R, C, m)$, where m is the signing message.
- Each signer calculates a partial signature (R, u_i, v_i) where $u_i = k_{1i} + e f_i x_i \bmod p$, and $v_i = k_{2i} + e f_i a_i \bmod p$.
- Combine all partial signatures to get the final signature (R, u, v) where $u = \sum_{i=1}^{n} u_i$ and $v = \sum_{i=1}^{n} v_i$.

For signature verification on (R, u, v), calculate the $C = \sum_{i=1}^{n}(f_i * C_i)$ where $f_i = Hash([C_1, C_2, \dots, C_n], C_i)$, calculate $e = Hash(R, C, m)$, compute Pedersen commitment $S = g^u h^v$, and accept the signature if $S = RC^e$.

3.3 Optimization for Multiple Inputs Single Signer

In case the signing is by one single signer, for example one is spending multiple UTXO which are owned by him/her self, above multi-signature signing procedure can be simplified, making use of a single random *R* instead of multiple, as described below.

- For multiple Pedersen commitment $C_i = g^{x_i} h^{a_i}$ where $i \in [0..n]$, n is the total number.
- Calculate $C = \sum_{i=1}^{n}(f_i * C_i)$, where $f_i = Hash([C_1, C_2, \dots, C_n], C_i)$.
- Calculate $x = \sum_{i=1}^{n}(f_i * x_i)$ and $a = \sum_{i=1}^{n}(f_i * a_i)$.
- Generates a random Pedersen commitment $R = g^{k_1} h^{k_2}$, calculates $e = Hash(R, C, m)$, where m is the signing message.
- Calculates signature (R, u, v) where $u = k_1 + ex \bmod p$, and $v = k_2 + ea \bmod p$.

4 Applications

This *ComSig* signature scheme makes it possible to design a blockchain transaction scheme without an interactive procedure, if the blockchain is using the Pedersen commitment as output, makes the transaction much faster and easier for end user, since now a single signer can complete the signature to prove the ownership of a Pedersen commitment output.

Another interesting application of this *ComSig* signature scheme could be an optional solution to solve the Bitcoin address reuse issue, which refers to the use of the same address for multiple transactions and therefore harms the privacy or even likely in violation of reasonable consumer protection laws. The prior solution is to use deterministic wallets which make it easy and safe to have many wallet addresses, but obviously it's not quite convenient for user to refresh his/her address to the sender and can't avoid the sender to pay into same address. With *Pay-to-PubkeyHash* transaction type in Bitcoin as an example, the original script is like this:

$$scriptPubKey: OP_{DUP}\ OP_{HASH160} < pubKeyHash >\ OP_{EQUALVERIFY}\ OP_{CHECKSIG}$$
$$scriptSig: < sig > < pubKey >$$

With this *ComSig* signature scheme, it can be revised as:

$$scriptPubKey: OP_{DUP}\ OP_{HASH160} < commitHash >\ OP_{EQUALVERIFY}\ OP_{CHECKSIG}$$
$$scriptSig: < sig > < commit >$$

then, the Bitcoin address can be freely reused, since the Pedersen commitment hide the actual Bitcoin address, and only need to let the committed value a be a secret for a payment, that secret can be sent to the transaction receiver by a method outside the blockchain. The pros of above said solution is the recipient privacy, and the cons are mainly at the *ComSig* signature has a bigger size than ECDSA or Schnorr signature, which has a form as (R, u, v) instead of (R, s) and therefore the $scriptSig$ will increase the transaction payload size with 32 bytes.

Since the increased 32-bytes payload is not trivial in current Bitcoin transaction, this *ComSig* signature scheme is not optimal for Bitcoin, but indeed for the similar transaction-output-based blockchain system, which is using Pedersen commitment in the transaction output, for example the Gotts.

Before discussion a possible application in Gotts, I would like to give a short review on an existing proposal about the non-interactive transaction for Mimblewimble, which was

proposed by 'GandalfThePink' (one user of the Grin community) [Gan19]. According to his/her proposal, a Mimblewimble non-interactive transaction, with for example one single input and one single output, may be designed as

$$C_a + excess = C_b + fee * H$$

Where

- $C_a = C'_a + s_a * G$ where $C'_a = v_a * H + k_a * G - s_a * G$, and k_a, s_a are Alice's private keys,
- $C_b = C'_b + s_b * G$ where $C'_b = v_b * H - s_b * G$, and S_b as Bob's private key with corresponding public key $P_b = s_b * G$, notes that missing '$k_b * G$' part in C'_b,
- $v_a = v_b + fee$, where fee is the transaction fee, and v_a, v_b are the transaction input/output amounts,
- $excess = s_b * G - k_a * G$, which is used as the public excess for signature.

Alice, as the sender who has an UTXO (C'_a, P_a), may create this transaction with Bob's public key P_b, to produce an output (C'_b, P_b) to the chain. And Alice must attach a partial signature with private key k_a, with respect to the combined public key, meaning the public excess, $s_b * G - k_a * G$. This is not a valid signature until Bob adds his signature with s_b and aggregates them, but anyone can verify this partial signature with a public key $P' = P_b - excess$.

As Gandalf pointed in his document, there are two flaws in above non-interactive transaction solution. The first one is, Bob's public key P_b might contain some term proportional of H component, meaning $P_b = O + s_b * G$ where the overflow part $O = v * H$ with $v > 0$, or creating new money out of the air. The second flaw is the double spent, that Alice is able to repeatedly spend the input (C'_a, P_a). Gandalf solved the first flaw by introducing the concept of a 'hanging' transaction and letting the general chain validation skip the 'hanging' outputs. And for second flaw, he proposed to consider an input as spent and record this on the chain as soon as only signatures to the public keys of the outputs are missing.

For Gandalf's solution, I doubt the major problem will be that there is no way for Alice to produce the range proof of output (C'_b, P_b) without the knowledge of Bob's private key s_b, even if ignoring the aspects of the complexity of the mixing 'hanging' transaction with normal Mimblewimble transaction. Comparing to Grin Mimblewimble interactive transaction solution with Schnorr signature scheme, this non-interactive transaction solution increases a public key, 48 bytes with a typical BLS signature scheme, to the transaction data, together with the increased size of BLS signature which typically needs 96 bytes. Therefore, each output will increase 48 bytes plus each transaction increases 32 bytes.

In Grin privacy transaction, this increased size is trivial since a typical transaction with single input and double outputs in Grin need about 1.6k bytes. But in Gotts, which removed the heaviest range proof part, that's not a trivial size increment, since a typical transaction in Gotts may be only 250 bytes.

Using the *ComSig* signature scheme, there should be a very simple non-interactive transaction solution for Gotts, since there's no requirement to have the range proof because Gotts supports the public value instead of the hidden value. In Gotts, a non-interactive transaction may be defined as:

$$(x_i * G + w_i * H) + excess' = (x_c * G + w_c * H) + (P_r + w_r * H),$$

$a_i = a_c + a_r + fee$, where a_i, a_c, a_r are the amounts; fee is the transaction fee,

Where,

- $(x_i * G + w_i * H)$ is the input commitment owned by sender,
- $(P_r + w_r * H)$ is the output commitment for receiver, where $P_r = x_r * G$ is receiver's public key with x_r as the private key,
- $(x_c * G + w_c * H)$ is the change commitment for sender,
- x_i, x_c are the sender's private keys,

- $w_i = w_c + w_r$, where w_i, w_c, w_r are the random numbers.

The $excess'$ is called as "*public excess*" which is no more the signature public key, and it consists of:

$$excess' = (x_c - x_i) * G + P_r$$

Where,

- $(x_c - x_i) * G$ is a public key which only sender knows the private key,
- P_r is a public key which only receiver knows the private key.

To sign this transaction, with the input commitment $(x_i * G + w_i * H)$ as the public key, to prove the ownership of it, the *ComSig* signature scheme is used. This completely removes the interactive communication requirement in Gotts, with the only cost of a bigger signature data, 96 bytes instead of 64 bytes. As an additional data, the w_r must be encoded as w_r' and attached to the transaction.

To discuss the security of this non-interactive transaction solution, let's imagine the receiver's public key P_r might contain some term proportional of H component, meaning $P_r = x_r * G + v * H$ where $v > 0$. In Gotts, this will never be related to money creation, since the amount is transparent and separated from the commitment balance equation $(x_i * G + w_i * H) + excess' = (x_c * G + w_c * H) + (P_r + w_r * H)$. Or in other words, this P_r is even purposely allowed to contain H component to protect the receiver's address privacy.

To encode the w_r, a naive solution may be $w_r' = w_r \; XOR \; Hash(P_r)$ if the receiver has a secured communication channel to tell the sender about this P_r, whereas a more practical and secure encoding method will be discussed in the Gotts stealth address document.

Reference

Sch90	Claus P. Schnorr. Schnorr Signature. (1990) U.S. Pat. No. 4,995,082.
DCC19	Gregory Maxwell, Andrew Poelstra, Yannick Seurin, Pieter Wuille. Simple Schnorr multi-signatures with applications to Bitcoin. Des. Codes Cryptogr. (2019) 87: 2139. https://doi.org/10.1007/s10623-019-00608-x.
Ped91	Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Crypt, volume 91, pages 129-140. Springer, 1991.
MD17	Roberto Metere, Changyu Dong. Automated Cryptographic Analysis of the Pedersen Commitment Scheme. MMM-ACNS 2017. Lecture Notes in Computer Science, vol 10446.
MRK03	S. Micali, M. Rabin, J. Kilian. Zero-knowledge sets. 44th Annual IEEE Symposium on Foundations of Computer Science, 2003. Proceedings.
RM17	Tim Ruffing, Giulio Malavolta. (2017) Switch Commitments: A Safety Switch for Confidential Transactions. In: Brenner M. et al. (eds) Financial Cryptography and Data Security. FC 2017. Lecture Notes in Computer Science, vol 10323.
Bit08	Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System, 2008. http://bitcoin.org/bitcoin.pdf
MW16	Tom Elvis Jedusor. Mimblewimble. 2016. https://github.com/mimblewimble/docs/wiki/MimbleWimble-Origin
Gri17	Ignotus Peverell. Introduction to Mimblewimble and Grin. 2017. https://github.com/mimblewimble/grin/blob/master/doc/intro.md
Lip03	Helger Lipmaa. Cryptography and data security lecture 7: ZK and commitments. 2003. http://www.tcs.hut.fi/Studies/T-79.159/2004/slides/L8.pdf
Li12	Ninghui Li. Cryptography lecture topic 23: Zero-Knowledge Proofs and Commitment Schemes. 2012. https://www.cs.purdue.edu/homes/ninghui/courses/555_Spring12/handouts/555_Spring12_topic23.pdf

Pon19 User ‘poncho’. Extension of the Schnorr protocol. 2019. https://crypto.stackexchange.com/a/67263/54007
Gan19 User ‘GandalfThePink’. BLS signatures in Mimblewimble. 2019. https://github.com/mimblewimble/grin/files/2905763/MWpp.pdf