Mimblewimble

Saravanan Vijayakumaran
sarva@ee.iitb.ac.in

Department of Electrical Engineering
Indian Institute of Technology Bombay

November 5, 2019

Mimblewimble

Mimblewimble, which prevents your opponent from accurately casting their next spell.

Gilderoy Lockhart

- A tongue-tying curse from the Harry Potter universe
- A scalable cryptocurrency design with hidden amounts and obscured transaction graph
- Brief history
  - **Aug 2016:** “Tom Elvis Jedusor” posted an onion link to a text file describing Mimblewimble on bitcoin-wizards IRC channel
  - **Oct 2016:** Andrew Poelstra presents formalization of Mimblewimble at Scaling Bitcoin 2016
  - **Oct 2016:** “Ignotus Peverell” announces a project implementing the Mimblewimble protocol called Grin
  - **Jul 2018**: Another Mimblewimble implementation called BEAM announced
  - **Jan 2019:** BEAM launched on Jan 3, 2019 and Grin launched on Jan 15, 2019

Mimblewimble Outputs

- Recall the structure of Monero outputs
  - A public key P acting as destination address
  - A Pedersen commitment C to the amount stored in the output
  - A range proof proving the amount in C is in the right range
- Mimblewimble output structure
  - A Pedersen commitment C where

    $$C = kG + vH$$

    where G and H are generators of an elliptic curve of prime order n and the discrete logarithm of H wrt G is unknown
  - A range proof proving the amount in C is in a range like $\{0, 1, 2, \ldots, 2^{64} - 1\}$
- Features of Mimblewimble output variables
  - The order n is typically a 256-bit prime, i.e. $n \approx 2^{256}$
  - The scalar $v \in \mathbb{F}_n$ is the amount
  - The scalar $k \in \mathbb{F}_n$ is the blinding factor (will play role of **secret key**)

Proving Statements About Commitments

- How to prove that C is a commitment to the zero amount without revealing blinding factor?

  Ans: If $C = C(0, x) = xG$, then give a digital signature verifiable by C as the public key

  If C is a commitment to a non-zero amount a, signature with C as public key will mean discrete log of H is known

  $$C = xG + aH = yG \implies H = a^{-1}(y - x)G$$

- How to prove that C is a commitment to the an amount a without revealing blinding factor?

  Ans: If $C = C(a, x) = xG + aH$, then give a digital signature verifiable by $C - aH$ as the public key

- How to prove that two commitments C_1 and C_2 are commitments to the same amount a without revealing blinding factors?

  Ans:

  $$\begin{aligned} C_1 &= C(a, x_1) = x_1 G + aH \\ C_2 &= C(a, x_2) = x_2 G + aH \end{aligned}$$

  Give a digital signature verifiable by $C_1 - C_2$ as the public key

Proving the Balance Condition

- Suppose $C_1^{\text{in}}, C_2^{\text{in}}, C_3^{\text{in}}$ are commitments to input amounts a_1, a_2, a_3
- Suppose $C_1^{\text{out}}, C_2^{\text{out}}$ are commitments to output amounts b_1, b_2
- Suppose we want to prove

$$a_1 + a_2 + a_3 = b_1 + b_2 + f$$

for some public $f \geq 0$

- A digital signature with

$$C_1^{\text{in}} + C_2^{\text{in}} + C_3^{\text{in}} - C_1^{\text{out}} - C_2^{\text{out}} - fH$$

as public key is enough

- **Almost enough!** It only shows that

$$\begin{aligned} & a_1 H + a_2 H + a_3 H = b_1 H + b_2 H + fH \\ \Longrightarrow \ & a_1 + a_2 + a_3 = b_1 + b_2 + f \bmod n, \end{aligned}$$

since $nH = \mathcal{O}$ (the identity of the elliptic curve group)

Preventing Exploitation of the Modular Balance Condition

$$a_1 + a_2 + a_3 = b_1 + b_2 + f \bmod n$$

- **Example:** $a_1 = 1, a_2 = 1, a_3 = 1$ and $b_1 = n - 4, b_2 = 6, f = 1$
- Typically $n \approx 2^{256}$ and amounts are in a smaller range like $\{0, 1, 2, \ldots, 2^{64} - 1\}$
- Proving that C_1^{out} and C_2^{out} commit to amounts in the range $\{0, 1, 2, \ldots, 2^{64} - 1\}$ solves the problem
- Each output should be accompanied by a range proof

Mimblewimble Transactions

- Each transaction has
  - L input commitments $C_1^{\text{in}}, C_2^{\text{in}}, \dots, C_L^{\text{in}}$
  - M output commitments $C_1^{\text{out}}, C_2^{\text{out}}, \dots, C_M^{\text{out}}$ with range proofs
  - N **transaction kernels**
  - A scalar $k_{\text{off}} \in \mathbb{F}_n$ called the **kernel offset**
- Each transaction kernel has the following
  - A scalar $f_i \in \mathbb{F}_n$ representing a fee
  - A curve point $X_i = x_i G$ called the **kernel excess**
  - A Schnorr signature verifiable with X_i as the public key
- For $f = \sum_{i=1}^{N} f_i$, the following equality is checked

$$\sum_{i=1}^{M} C_i^{\text{out}} + fH - \sum_{i=1}^{L} C_i^{\text{in}} = \sum_{i=1}^{N} X_i + k_{\text{off}} G$$

- This ensures

$$\sum_{i=1}^{L} v_i^{\text{in}} = \sum_{i=1}^{M} v_i^{\text{out}} + f \quad \text{and} \quad \sum_{i=1}^{M} k_i^{\text{out}} - \sum_{i=1}^{L} k_i^{\text{in}} = \sum_{i=1}^{N} x_i + k_{\text{off}}$$

- The offset k_{off} is used to hide relationship between specific inputs and outputs of a transaction during block creation

Schnorr Signature Algorithm

- Let G be a cyclic group of order q with generator g
- Let $H : \{0,1\}^* \mapsto \mathbb{Z}_q$ be a cryptographic hash function
- Signer knows $x \in \mathbb{Z}_q$ such that public key $h = g^x$
- **Signer:**
  1. On input $m \in \{0,1\}^*$, chooses $r \leftarrow \mathbb{Z}_q$
  2. Sets $I := g^r$
  3. Computes $e := H(I, m)$
  4. Computes $s = ex + r \bmod q$
  5. Outputs (e, s) as signature for m
- **Verifier**
  1. On input m and (e, s)
  2. Compute $I := g^s \cdot h^{-e}$
  3. Signature valid if $H(I, m) \stackrel{?}{=} e$

Mimblewimble Transaction Construction

- Unlike other cryptocurrencies, sender and receiver have to **interact** to construct a Mimblewimble transaction
- Interaction can be via email, chat, forum posts
- Suppose Alice owns unspent output $C_{\text{in}} = k_A G + v_A H$
- She wants to send v_B coins to Bob where $v_B < v_A$
- She will be paying transaction fees f
- She wants the remaining $v_A - v_B - f$ coins to be stored in a change output $C_{\text{chg}} = k_C G + (v_A - v_B - f)H$
- Bob wants his new output to have blinding factor k_B, i.e. $C_{\text{out}} = k_B G + v_B H$
- Alice and Bob will exchange a data structure called a **slate**
- **Step 1**
  - Alice adds C_{in}, amount v_B, fees f to the slate
  - She chooses $k_C \xleftarrow{\$} \mathbb{F}_n$, calculates $C_{\text{chg}} = k_C G + (v_A - v_B - f)H$ and a range proof
  - She chooses kernel offset $k_{\text{off}} \xleftarrow{\$} \mathbb{F}_n$ and calculates the **sender kernel excess secret key** as $k_A' = k_C - k_A - k_{\text{off}}$
  - k_{off} and the **sender kernel excess** $X_A = k_A' G$ are added to the slate
  - She chooses nonce $r_A \xleftarrow{\$} \mathbb{F}_n$ and adds the nonce public key $R_A = r_A G$ to the slate.
  - Alice sends slate to Bob

Mimblewimble Transaction Construction

- **Step 2**
  - Bob chooses $k_B \xleftarrow{\$} \mathbb{F}_n$, calculates $C_{\text{out}} = k_B G + v_B H$ and a range proof. He adds C_{out} to the slate.
  - He adds k_{off} and the **receiver kernel excess** $X_B = k_B G$ are added to the slate
  - He chooses nonce $r_B \xleftarrow{\$} \mathbb{F}_n$ and adds the nonce public key $R_B = r_B G$ to the slate.
  - Bob calculates the receiver Schnorr signature on message m as (s_B, R_B) where $s_B = r_B + ek_B$ and

    $$e = \text{SHA256}(R_A + R_B \| X_A + X_B \| m).$$

    He adds the signature to the slate. It can be verified using the public key X_B.
  - Bob sends slate to Alice
- **Step 3**
  - Alice verifies Bob's signature (s_B, R_B) by checking the equality

    $$s_B G = R_B + eX_B,$$

  - She calculates the sender Schnorr signature (s_A, R_A) on the same message m as $s_A = r_A + ek'_A$
  - She sets the transaction kernel excess to be equal to $X_A + X_B$.
  - She sets the signature in the transaction kernel to be equal to $(s_A + s_B, R_A + R_B)$.

Mimblewimble Transaction Construction

- Alice broadcasts transaction k_{off}, C_{in}, C_{out}, C_{chg}, and the transaction kernel
- Kernel contains fee f, the kernel excess $X_A + X_B$, and the signature $(s_A + s_B, R_A + R_B)$
- Transaction satisfies

$$\begin{aligned}
&C_{\text{out}} + C_{\text{chg}} + fH - C_{\text{in}} \\
&= \; k_B G + v_B H + k_C G + (v_A - v_B - f)H + fH - k_A G - v_A H \\
&= \; k_B G + (k_C - k_A)G \\
&= k_B G + (k_C - k_A - k_{\text{off}})G + k_{\text{off}} G \\
&= \; k_B G + k'_A G + k_{\text{off}} G = X_B + X_A + k_{\text{off}} G.
\end{aligned}$$

- Alice does not learn Bob's blinding factor k_B
- Bob learns neither change amount $v_A - v_B - f$ nor blinding factor k_C

References

- Mimblewimble original paper
  https://scalingbitcoin.org/papers/mimblewimble.txt
- A short history of Mimblewimble https://medium.com/beam-mw/
  a-short-history-of-mimblewimble-from-hogwarts-to-mobile-wallets-2
- Poelstra talk in BPASE 2017 https://cyber.stanford.edu/sites/g/
  files/sbiybj9936/f/andrewpoelstra.pdf
- Grin GitHub repo https://github.com/mimblewimble/grin
- BEAM website https://beam.mw/
- Intro to Mimblewimble and Grin https:
  //github.com/mimblewimble/grin/blob/master/doc/intro.md
- BEAM announcement
  https://medium.com/beam-mw/introducing-beam-f35096a923ec