IT486 v3.0: Blockchains and Cryptocurrencies

Signature aggregation, Bitcoin governance

ECDSA

- What Bitcoin, other cryptocurrencies use today
- Made to avoid a patent on a better signature system
- That patent has expired, we are free to use the simpler better algo

EC Schnorr

- Have message m, private key k, public key $P = kG$
- Make secret nonce r, public key of nonce $R = rG$
- Challenge: $e = H(R|P|m)$
- Signature: $s = r + ek$

EC Schnorr

- Have message m, private key k, public key $P = kG$
- Make secret nonce r, public key of nonce $R = rG$
- Challenge: $e = H(R|P|m)$
- Signature: $s = r + ek$
- Verification: $sG == R + eP$

EC Schnorr

- Have message m, private key k, public key $P = kG$
- Make secret nonce r, public key of nonce $R = rG$
- Challenge: $e = H(R|P|m)$
- Signature: $s = r + ek$
- Verification: $sG == R + eP$
- Anyone can verify the signature from R and P

EC Schnorr

- Let's say we sign a message m with $e = H(P|m)$
- Signature: $s = ek$
- Now we can check the signature is valid: $sG == eP$

EC Schnorr

- Let's say we sign a message m with $e = H(P|m)$
- Signature: $s = ek$
- Now we can check the signature is valid: $sG == eP$
- But anyone can read your private key now; all they have to do is solve $k = s/e$

EC Schnorr

- Let's say we sign a message m with $e = H(P|m)$
- Signature: $s = ek$
- Now we can check the signature is valid: $sG == eP$
- But anyone can read your private key now; all they have to do is solve $k = s/e$
- With the nonce r you have to solve $k = (s - r)/e$, but r is unknown

Naive Signature Aggregation

- Let's say Alice and Bob want to sign the same message m
- They have private keys k_a and k_b and public keys P_a and P_b
- They have secret nonces r_a and r_b and public keys of their nonces $R_a = r_a G$ and $R_b = r_b G$

Naive Signature Aggregation

- Let's say Alice and Bob want to sign the same message m
- They have private keys k_a and k_b and public keys P_a and P_b
- They have secret nonces r_a and r_b and public keys of their nonces $R_a = r_a G$ and $R_b = r_b G$
- Make a shared public key $P_{agg} = P_a + P_b$
- Challenge: $e = H(R_a|R_b|P_a|P_b|m)$
- Joint signature: $s_{agg} = s_a + s_b$

Naive Signature Aggregation

- Let's say Alice and Bob want to sign the same message m
- They have private keys k_a and k_b and public keys P_a and P_b
- They have secret nonces r_a and r_b and public keys of their nonces $R_a = r_a G$ and $R_b = r_b G$
- Make a shared public key $P_{agg} = P_a + P_b$
- Challenge: $e = H(R_a|R_b|P_a|P_b|m)$
- Joint signature: $s_{agg} = s_a + s_b$
- Verification: $s_{agg} G == (R_a + R_b) + e(P_a + P_b)$

Naive Signature Aggregation

- Let's say Alice and Bob want to sign the same message m
- They have private keys k_a and k_b and public keys P_a and P_b
- They have secret nonces r_a and r_b and public keys of their nonces $R_a = r_a G$ and $R_b = r_b G$
- Make a shared public key $P_{agg} = P_a + P_b$
- Challenge: $e = H(R_a|R_b|P_a|P_b|m)$
- Joint signature: $s_{agg} = s_a + s_b$
- Verification: $s_{agg} G == (R_a + R_b) + e(P_a + P_b)$
- Note: Anyone can verify the joint signature from the sum of the Rs and Ps

Key Cancellation Attack

- Bob knows P_a and R_a ahead of time
- Bob says his public key is $P'_b = P_b - P_a$ and the public key of his nonce is $R'_b = R_b - R_a$
- Note: Bob doesn't know the private keys for these faked values

Key Cancellation Attack

- Bob knows P_a and R_a ahead of time
- Bob says his public key is $P'_b = P_b - P_a$ and the public key of his nonce is $R'_b = R_b - R_a$
- Note: Bob doesn't know the private keys for these faked values
- Everyone assumes $S_{agg}G = R_a + R'_b + e(P_a + P'_b)$
- But then $s_{agg}G = r_bG + ek_bG$, so $s_{agg} = s_b$

Key Cancellation Attack

- Bob knows P_a and R_a ahead of time
- Bob says his public key is $P'_b = P_b - P_a$ and the public key of his nonce is $R'_b = R_b - R_a$
- Note: Bob doesn't know the private keys for these faked values
- Everyone assumes $S_{agg}G = R_a + R'_b + e(P_a + P'_b)$
- But then $s_{agg}G = r_bG + ek_bG$, so $s_{agg} = s_b$
- Bob can create this signature himself!

Attack prevention

- In the attack, Bob didn't know the private keys for his published R and P values

Attack prevention

- In the attack, Bob didn't know the private keys for his published R and P values
- We can defeat Bob by asking him to sign a message proving that he does know the private keys

MuSig scheme

- Alice and Bob have private-public key pairs (k_a, P_a) and (k_b, P_b)
- They publish the public keys of their nonces, R_a and R_b
- They calculate a shared public key P as follows:

$$
\begin{aligned}
l &= H(P_a|P_b) \\
a_i &= H(l|P_i), i \in \{a, b\} \\
P &= \sum a_i P_i
\end{aligned}
$$

MuSig scheme

- They calculate a shared nonce, $R = R_a + R_b$
- The challenge $e = H(R|P|m)$
- Each signer provides their contribution to the signature as:

  $$s_i = r_i + a_i e k_i$$

- Join signature: $s = \sum s_i$
- Verification: $sG == R + eP$

Security of MuSig

- Suppose Bob provides fake values for his nonce and public key:

$$R_f = R_b - R_a$$
$$P_f = P_b - P_a$$

- This leads to both Alice and Bob calculating the following shared values:

$$I = H(P_a|P_f)$$
$$a_a = H(I|P_a)$$
$$a_f = H(I|P_f)$$
$$P = a_a P_a + a_f P_f$$
$$R = R_a + R_f$$
$$e = H(R|P|m)$$

Security of MuSig

- Bob then tries to construct a unilateral signature:

  $$s_b = r_b + k_s e$$

- For the attack to succeed, Bob needs to find k_s such that $s_b G == R + eP$

Security of MuSig

- Bob then tries to construct a unilateral signature:

  $$s_b = r_b + k_s e$$

- For the attack to succeed, Bob needs to find k_s such that $s_b G == R + eP$
- But this is not a feasible calculation – why?

Security of MuSig

$$
\begin{aligned}
s_b G &= R + eP \\
(r_b + k_s e)G &= R_b + e(a_a P_a + a_f P_f) \\
&= R_b + e(a_a P_a + a_f P_b - a_f P_a) \\
&= (r_b + e a_a k_a + e a_f k_b - e a_f k_a)G \\
k_s e &= e a_a k_a + e a_f k_b - e a_f k_a \\
k_s &= a_a k_a + a_f k_b - a_f k_a
\end{aligned}
$$

Bitcoin governance process

- Software needs to be changed from time to time:
- For centrally controlled software (e.g. Microsoft, Google, Facebook) changed can be effected unilaterally

Bitcoin Core

- Bitcoin Core software is open source
- Maintainers control repository
- Maintainers determined by agreement of community
- Change proposals documented for discussion as Bitcoin improvement proposals (BIP's) - similar to Internet RFP's