Instantiating Scriptless 2P-ECDSA

Fungible 2-of-2 Multisigs for Today's Bitcoin

Conner Fromknecht

Head of Cryptographic Engineering, Lightning Labs

History and Motivation

- Andrew Poelstra releases work on Schnorr-based Scriptless Scripts in 2016
    - Follows up w/ construction for replacing hash-preimage challenge in LN (March 2017)
    - Ensures payment information is randomized at every hop
        - Increases privacy
        - Prevents fee-siphoning mentioned by Pedro
- In late 2017, Yehuda Lindell publishes efficient 2P-ECDSA signing protocol
    - Offers ability to do 2-of-2 ECDSA multisigs **without updating consensus layer**
    - Retains anonymity set of existing P2WKH wallet transactions
- April 2018, Multi-Hop Locks paper delivers formalized framework for LN hop decorrelation
    - Includes Scriptless Script construction based on 2P-ECDSA protocol
    - Shows that Schnorr and 2P-ECDSA are fully interoperable
        - Can be mixed heterogeneously on same LN path!
        - Implies that switching from ECDSA to Schnorr **won't fragment the network**

LN gains of moving to Schnorr/ECDSA Multi-Hop Locks:

- Increased privacy for both on-chain and off-chain transactions
- Smaller transactions and fees
- *Real* proofs-of-payment ("Invoice Tunneling")
- More extensions yet to come ;)

Agenda

I. 2P-ECDSA Overview

II. Benchmarks

III. Applications to Lightning

IV. Deployment Considerations

2P-ECDSA Overview

Two participants, Alice and Bob:

- Alice has private key *a* and public key *A = a*G*
- Bob has private key *b* and public key *B = b*G*
- Jointly create public key *Q = ab*G* with private key *ab*, but...
  - **neither knows** *ab* outright, yet...
  - together they can create valid ECDSA signatures under *Q*

Requires two algorithms:

- **`KeyGen`** (offline):
  - Sets up Alice and Bob for participation in online signing protocol
  - More expensive, but only executed once
- **Sign** (online):
  - Produces an ECDSA signature under *Q*
  - 2 RTT

- Fast Secure Two-Party ECDSA Signing, Yehuda Lindell. https://eprint.iacr.org/2017/552.pdf
- Efficient RSA Key Generation and Threshold Paillier in the Two-Party Setting, Hazay et. al. https://eprint.iacr.org/2011/494.pdf
- A Generalisation, a Simplification and some Applications of Paillier's Probabilistic Public-Key System, Damgard and Jurik. http://www.brics.dk/RS/00/45/BRICS-RS-00-45.pdf

2P-ECDSA Overview (KeyGen)

- **`KeyGen`**:
  - Alice and Bob exchange pubkeys, each provides discrete log PoK
  - Alice generates a Paillier keypair *(PSK, PPK)*
    - Provides ZKP that *PPK* is constructed properly, $N = p_1 * p_2$
  - Alice encrypts her private key under PPK, creating ciphertext $c = Enc_{PPK}(a)$
  - Alice sends *(PPK, c)* to Bob
    - Provides ZKP that ciphertext contains “small” value
      - i.e., $0 < Dec_{PSK}(c) < q$, where *q* is the secp256k1 curve order
    - AND that *c* contains Alice’s private key, $A = Dec_{PSK}(c) * G$
    - Lindell had to invent a new ZKP to do so!
  - Bob verifies all the proofs and computes $Q = b * A$
  - Alice computes $Q = a * B$
  - Output:
    - Alice saves 2P-ECDSA private key *(a, PSK)* with public key $Q = ab*G$
    - Bob saves 2P-ECDSA private key *(b, c, PPK)* with public key $Q = ab*G$

2P-ECDSA Overview (Sign)

- Sooo why all this Paillier nonsense?
  - Can't "add" signatures and pubkeys as we can with Schnorr
  - Paillier ciphertexts exhibit partially-homomorphic properties
    - Additive: $D(E(m1) * E(m2) \bmod N^2) = m1 + m2 \bmod N$
    - Scalar-multiplicative: $D(E(m)^k \bmod N^2) = k*m \bmod N$
    - Both can be done **without private knowledge**

- ECDSA signature: (R, s) where $s = k^{-1} * (H(m) + r * x)$ and $r = x\text{-coord}(R = k*G)$

- **Sign**:
  - Alice and Bob exchanges nonces w/ discrete log PoK, $K_a = k_a*G$ and $K_b = k_b*G$
  - Bob encrypts $c_1 = Enc_{PPK}(k_b^{-1} * H(m))$ and $v = k_b^{-1} * r * b$ where $r = x\text{-coord}(R = k_a^{-1}k_b^{-1} * G)$
  - Bob computes and sends $\boldsymbol{c' = c_1 * c^\wedge v \bmod N^\wedge 2} = Enc_{PPK}(k_b^{-1} * (H(m) + r * a * b))$ to Alice
  - Alice $s' = Dec_{PSK}(c')$ and computes $s'' = k_a^{-1} * s' \bmod q$
  - Sure enough, $s'' = k_a^{-1} * k_b^{-1} * (H(m) + r * a * b)$
  - Finally, Alice sets $s = \min(s'', q - s'' \bmod q)$ and outputs signature (R, s)

Benchmarks

	Time	Memory Allocated	Num Allocations	Num Messages
KeyGen[1]	1.07 s	4.99 MB	13.31 K	7
Sign	28.66 ms	97 KB	746	4
Scriptless-Sign	29.40 ms	118 KB	1.12 K	5+1

Setup:

- 2.8 GHz Intel Core i7 16 GB 2133 MHz LPDDR3
- Single process, no network latency or serialization
- Non-interactive DLOG PoK and Proof of Paillier Paillier Key Correctness
- Interactive Paillier Range and DLog Ciphertext Proof

Golang code will be published here: https://github.com/cfromknecht/tpec (1.7K LOC)
Concurrent work in Rust by Gary Bennatar and Omer Shlomovits: https://github.com/KZen-networks/multi-party-ecdsa

- Also working on t-of-n threshold ECDSA signing!

[1] Likely to change after further refinement and optimization

Deployment Considerations - Script Modifications

- 2P-ECDSA/Schnorr
  - Funding Outputs
    - Currently regular 2-of-2 multisigs
    - Requires 2-of-2 signature to spend
      - Cooperative closes
      - Commitment transactions
    - Replaced with P2WKH-looking output
  - HTLC Outputs
    - Uses 2-of-2 multisig in non-standard HTLC scripts
    - Two types of HTLC scripts: offered and received
    - Requires 2-of-2 sig to spend offered-timeout and received-success clauses
    - Replaced with *much* simpler HTLC script

- Scriptless 2P-ECDSA/Schnorr
  - HTLC Outputs
    - Remove payment hashes from HTLC scripts!
    - By extension, remove preimages from witnesses

Deployment Considerations - Funding Output Scripts

	Witness	Witness Script
Regular 2-of-2 MultiSig	`OP_0 <a*G sig> <b*G sig>`	`OP_2 <a*G pubkey> <b*G pubkey> OP_2 OP_CHECKMULTISIG`
Schnorr 2-of-2 MultiSig	`<(a+b)*G sig>`	`<(a+b)*G pubkey> OP_CHECK_SCHNORR_SIG`
2P-ECDSA 2-of-2 MultiSig	`<ab*G sig>`	`<ab*G pubkey> OP_CHECKSIG`
P2WKH	`<a*G sig>`	`<a*G pubkey> OP_CHECKSIG`

- Witness bytes required to spend:
  - Regular 2-of-2: ~220
  - 2P-ECDSA and P2WKH: ~109
  - Schnorr: 100
- 2P-ECDSA is indistinguishable from P2WKH, increased anonymity set
  - Huge win for non-advertised channels

Deployment Considerations - HTLC Scripts

New Received-HTLC Witness Script

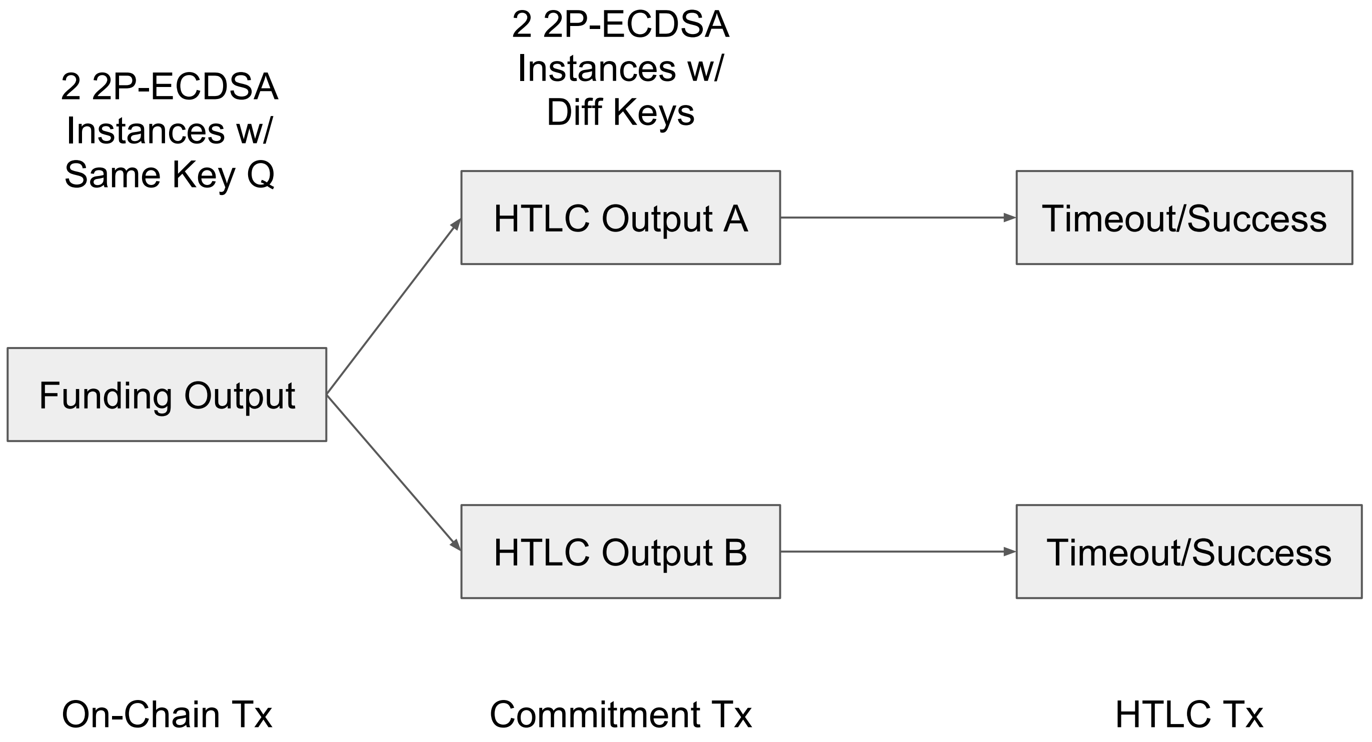
```
OP_IF
  # revocation clause
  revocationpubkey
OP_ELSE
  OP_IF
    # timeout clause
    <cltv_expiry> OP_CLTV OP_DROP
    <remote_delay_pubkey>
  OP_ELSE
    # success clause
    <2p_htlc_pubkey>
  OP_ENDIF
OP_ENDIF
OP_CHECKSIG
```

- 20% reduction in witness script size
- Improves readability immeasurably
- Reduced witness size
  - 78% for success witness
  - 30% for revocation witness
  - Timeout witness stays the same

Current Received-HTLC Witness Script

```
# revocation clause
OP_DUP OP_HASH160
<RIPEMD160(SHA256(revocationpubkey))> OP_EQUAL
OP_IF
  OP_CHECKSIG
OP_ELSE
  <remote_htlcpubkey> OP_SWAP OP_SIZE 32 OP_EQUAL
  OP_IF
    # success clause
    OP_HASH160 <RIPEMD160(payment_hash)>
OP_EQUALVERIFY
    2 OP_SWAP <local_htlcpubkey> 2 OP_CHECKMULTISIG
  OP_ELSE
    # timeout clause
    OP_DROP <cltv_expiry> OP_CLTV OP_DROP
    OP_CHECKSIG
  OP_ENDIF
OP_ENDIF
```

Deployment Considerations - 2P-ECDSA Instances



Deployment Considerations - Onion Packets

BOLT 04 Onion Packet Structure

1:realm	x:per-hop	32:MAC	19*(33+x):filler

Current `per-hop` payload (32 bytes)

```
[8:short_channel_id]
[8:amt_to_forward]
[4:outgoing_cltv_value]
[12:padding]

Total: 1300 bytes
```

MHL `per-hop` payload (161 bytes)

```
[8:short_channel_id]
[8:amt_to_forward]
[4:outgoing_cltv_value]
[33:incoming_lock_pubkey]
[64:incoming_lock_dlog_pok
[32:hop_lock_secret]
[12:padding]
Total: 3880 bytes
```

- Performance is dominated by asymmetric operations
  - Used to derive per-hop ephemeral keys and blinding factors
  - But, scales linearly in the number of hops!
- Increased message size likely have marginal impact on construction/decryption

Thank You!