Efficiently Batch Generating N-of-N ECDSA Threshold Single Signatures in a Semi-Honest Model and Uses for Bitcoin

Jeremy Rubin

October 23, 2018

Abstract

Semi-Honest batch generated N-of-N ECDSA threshold single signatures are a useful new tool for building backwards-compatible smart contracts for Bitcoin.

Protocols for N-of-N threshold ECDSA have been demonstrated previously, but are usually inefficient because of requirements for expensive consistency checks which add new assumptions and complexity to these protocols. This protocol is unique because it does not require any consistency checks by working a semi-honest model sufficient for a broad class of applications.

Non backwards compatible schemes – such as Schnorr Signatures, which requires modifications to Bitcoin – have also been demonstrated previously, but because they require changes they are not guaranteed to be available or accepted.

1 Assumptions

1.1 Elliptic Curves

Assume a curve with a base point denoted $G = (x_G, y_G)$ of order N (e.g. Secp256k1).

1.2 Semi-Honest Oblivious Transfer Multiplication

We demonstrate a protocol whereby two participants with a secret a and b, can each learn a new secret share τ_a and τ_b such that $\tau_a + \tau_b = a \cdot b$ without either party learning anything about a, b, or ab. If the protocol fails, both parties may learn a and b. if the protocol succeeds, only τ_a, τ_b are learned.

We denote this scheme $\pi_*(a, b) = ab = \tau_a + \tau_b$.

We show how such a scheme can be built from a 1 of 2 Oblivious Transfer assumption. HL17 presents an efficient scheme for 1-of-N Oblivious Transfer under the Computational Diffie Helman assumption.

1.3 Efficient OT Multiplication Protocol

Alice with an d-bit α, and a radix $r|d$, computes $\forall j \in 0 \cdots 2^d - 1.c_j \equiv a \times j \mod N$.

Alice then picks a set of $\frac{d}{r}$ blinding factors $\phi_i \xleftarrow{R} (0, N)$.

Alice's secret share $\sigma_a \equiv \sum_{i=0}^{r-1} \phi_i \mod N$

Alice then computes $\forall i \in [0, \frac{d}{r}).\forall j \in [0, 2^{\frac{d}{r}}).x_{i,j} \equiv c_j 2^{i\frac{d}{r}} - \phi_i \mod N$. In other words, $x_{i,*}$ is a table of the potential blinded results of multiplying a by all possible $\frac{d}{r}$-bit values shifted to the correct magnitude.

For each row of this table, Alice does a $1 - of - 2^{\frac{d}{r}}$ Oblivious Transfer.

If $r = d$, Alice can perform a direct 1-of-2 OT. With $\frac{d}{r} > 1$, and no assumption of $1 - of - N$ OT, it may make sense to use a combination of keys to perform the transfer as shown in the Appendix A with some additional ciphertext overhead. HL17 provides 1-of-N OT, which may be used in place of it.

Bob, then inputs his value β broken up into $\frac{d}{r}$-bit groups into the OT protocol and learns and sums the results $\sigma_b \equiv \sum_{i=0}^{r-1} x_{i,\beta_i} \mod N$.

Thus,

$$\sigma_a + \sigma_b = \alpha \cdot \beta$$

1.4 Optimal Parameters

1.4.1 1-of-2 OT

An optimal selection of parameters for this protocol is $r = d$. Switching to a different radix, say $r = 32$, would result in the Oblivious Transfer of $8 * 32$ 256 bit keys which is identical to doing 256 256-bit Oblivious Transfers. but with a larger ciphertext to share. Indeed, for any r the number of Oblivious Transfers required is constant. Thus, concretely:

$$x = \begin{bmatrix} 0 & \alpha \\ \vdots & \vdots \\ 0 & \alpha 2^i \\ \vdots & \vdots \\ 0 & \alpha 2^{255} \end{bmatrix} - \begin{bmatrix} \phi_0 \\ \vdots \\ \phi_i \\ \vdots \\ \phi_{255} \end{bmatrix}$$

And Bob learns, through oblivious transfer:

$$\sigma_\beta = \left(x \odot \begin{bmatrix} \beta_0 = 0 & \beta_0 = 1 \\ \vdots & \vdots \\ \beta_i = 0 & \beta_i = 1 \\ \vdots & \vdots \\ \beta_{255} = 0 & \beta_{255} = 1 \end{bmatrix} \right) \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

The cost of this is $256 * 256 = 65536$ Oblivious Transfers, and approximately $256 * 256 * 2$ bits, or $16KiB$.

1.4.2 1-of-N OT

With an optimized 1-of-N OT which transfers 256 Bit Keys, like HL17, we are free to pick a more agressive radix such as $r = 32$. The general equation is that there are r oblivious transfers and $2^{\frac{256}{r}} \cdot r \cdot 32$ constant cost of bandwidth in bytes. Thus the cost of $r = 32$ is 32 Oblivious Transfers with the sending of $256 * 32 * 32$ bytes to transfer, or $\approx 256KiB$.

$$x = \alpha \begin{bmatrix} 0 & 1 & \cdots & 255 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 2^{8i} & \cdots & 255 \cdot 2^{8i} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 2^{248} & \cdots & 255 \cdot 2^{248} \end{bmatrix} - \begin{bmatrix} \phi_0 \\ \vdots \\ \phi_i \\ \vdots \\ \phi_{31} \end{bmatrix}$$

And Bob learns, through oblivious transfer:

$$\sigma_b = \left(x \odot \begin{bmatrix} \beta_0 = 0 & \beta_0 = 1 & \cdots & \beta_0 = 255 \\ \vdots & \vdots & & \\ \beta_i = 0 & \beta_i = 1 & \cdots & \beta_i = 255 \\ \vdots & \vdots & \ddots & \vdots \\ \beta_{31} = 0 & \beta_{31} = 1 & \cdots & \beta_{31} = 255 \end{bmatrix} \right) \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}$$

2 Protocol

The signature algorithm has three components, Reusable Nonce Generation, Single Use Key Generation, and Signing.

2.1 Reusable Nonce Generation

Our goal is to generate a nonce K such that all participants learn $K \times G$ without learning K and each party learns a secret share μ_i such that $K^{-1} = \sum_{i=1}^{n} \mu_i$.

Let each party generate a secret k_i.

$$k_i \xleftarrow{R} [1, N-1]$$

Each party also computes the multiplicative modular inverse of k_i, q_i.

$$q_i k_i \equiv 1 \mod N$$

All parties compute the group element $K \times G$ corresponding to the nonce without revealing their nonce to anyone. This computation is done in a ring as demonstrated below.

$$(x_0, y_0) = K \times G = \left(\prod_{i=1}^{n} k_i\right) \times G = \left(\prod_{i=2}^{n} k_i\right) k_1 \times G = \left(\prod_{i=3}^{n} k_i\right) k_2 k_1 \times G = \ldots$$

Precommits are not needed in this ring compution because breaking the nonce generation would reduce to a break in ECDLH assumption. An adverasy at k_n has the ability to change $K \times G$ to a value of their choice. A later consistency check would fail were this done.

The public nonce r is computed as follows $r = x_0 \mod N$.

2.1.1 Reusability and Privacy

Additional nonces may be cheaply generated by participants by generating securely among all participants (perhaps using an efficient protocol like fair coin flipping over the phone) a second nonce $k'q' = 1 \mod N$ and multiplying $k'(K \times G)$ to get r' and a single multiplicative share of Q by q'. This assists in improving privacy of the protocol, but does not allow for keys to be reused. There should not be an issue with the fact that a factor of the nonce is known. Trivially, half of nonces are factorable by 2, just knowing 2 and $2^{-1} \mod N$ does not allow an adversary to break half of signatures.

This blinds the nonce from the public network which reduces linkability between transactions using the same nonce.

2.2 Single Use Key Generation

First, all parties generate a public key private key pair denoted.

Let

$$\alpha_i \xleftarrow{R} [1, N-1]$$

Let

$$\beta_i = \alpha_i \times G$$

Next, parties sum their public keys to produce an aggregate public key B.

Let

$$B = \sum_{i=1}^{n} \beta_i = \left(\sum_{i=1}^{n} \alpha_i\right) \times G$$

It is critical that every β_i is precommitted to before beginning the sum. Otherwise, an adverary may select $\beta_{adv} = \alpha_{adv} \times G - \beta_i$ to cancel out the ith key.

2.2.1 MuSig Key Aggregation

It is also possible to use MuSig Key Aggregation as follows:

Let

$$\alpha_i \xleftarrow{R} [1, N-1]$$

Let

$$\beta_i = \alpha_i \times G$$

Let

$$L = H(\beta_1 || \ldots || \beta_n)$$

Let

$$h_i = H(L || \beta_i)$$

Let

$$B = \sum_{i=1}^{n} \beta_i \cdot h_i$$

Let

$$\alpha_i' = \alpha_i \cdot h_i$$

This works in the plaintext model which eliminate extra rounds.

The security of this construction is in the MuSig paper.

The other benefit of this is that the keys would be compatible with Schnorr signatures once merged.

2.3 Semi-Honest Single Signing

Our goal is to compute the standard ECDSA signature equation

$$\sigma = (r, s) = (r, K^{-1}(M + r \cdot A) \mod N)$$

without any party learning K, A, or any other shares of K or A against a semi-honest adversary for a known message M. In our semi-honest model successfully generating σ – such that σ verifies as a standard-conforming ECDSA against the public key B for message M – guarantees the adversary was honest and no information was leaked, and failing to finish the protocol for any reason may leak the keys.

We set the desired message to be signed as $M = H(B? || \ldots)$. This is to emphasize that the message may be arbitrary, and it may need to include B (thereby excluding the use of public key recovery as an alternative to this scheme).

First we substitute the shares of

$$K^{-1} \equiv Q \equiv q_1 \cdots q_n \mod N$$

and of the secret key $A = \alpha_1 + \cdots \alpha_n \mod N$ into our equation.

$$s \equiv (q_1 \cdots q_n) \cdot (M + r \cdot (\alpha_1 + \ldots + \alpha_n)) \mod N$$

To simplify this expression, we distribute the r value over the sum of α_i secret shares and isolate just the first share (i.e., the protocol leader's) to sum with the message. Note that this reduction implies that the leader can arbitrarily select the message.

$$s \equiv (q_1 \cdots q_n) \cdot (M + (r \cdot \alpha_1 + \ldots + r \cdot \alpha_n)) \mod N$$

$$s \equiv (q_1 \cdots q_n) \cdot ((M + r \cdot \alpha_1) + r \cdot \alpha_2 + \ldots + r \cdot \alpha_n) \mod N$$

For convenience, we rewrite the right-hand factors as secrets η_i.

$$s \equiv (q_1 \cdots q_n) \cdot (\eta_1 + \ldots + \eta_n) \mod N$$

Then, we distribute a nonce share q_i over the nonce-blinded secret key terms.

$$s \equiv (q_2 \cdots q_n) \cdot (q_1\eta_1 + \ldots + q_1\eta_n) \mod N$$

And replace each individual operation with an oblivious transfer multiplication.

$$s \equiv (q_2 \cdots q_n) \cdot (\pi_*(q_1, \eta_1) + \ldots + \pi_*(q_1, \eta_n)) \mod N$$

We can optimize the term when $i = j$:

$$s \equiv (q_2 \cdots q_n) \cdot (q_1\eta_1 + \pi_*(q_1, \eta_2) + \ldots + \pi_*(q_1, \eta_n)) \mod N$$

Running the Oblivious Transfer Multiplication operations results in $2n - 1$ terms, denoted $\gamma_{i,j}$ for the term the ith participant learns from the interaction with the jth participant.

$$s \equiv (q_2 \cdots q_n) \cdot (\gamma_{1,1} + (\gamma_{1,2} + \gamma_{2,1}) + \ldots + (\gamma_{1,n} + \gamma_{n,1})) \mod N$$

This can be reduced back down to n terms.

$$s \equiv (q_2 \cdots q_n) \cdot ((\gamma_{1,1} + \gamma_{1,2} + \cdots + \gamma_{1,n}) + \gamma_{2,1} + \ldots + \gamma_{n,1}) \mod N$$

$$s \equiv (q_2 \cdots q_n) \cdot (\gamma_1 + \gamma_{2,1} + \ldots + \gamma_{n,1}) \mod N$$

After iterating the above step with every factor q_i, we are left with each participant knowing a single term Γ_i.

The participants reveal their Γ_i to the other participants, and all parties compute the sum.

$$\Gamma \equiv \Gamma_1 + \ldots + \Gamma_n \mod N$$

Revealing Γ_i plaintext is information theoretically secure because there are $n + 1$ terms known to each participant uniquely ($n - 1$ oblivious transfer multiplication shares γ, 1 q_i, and 1 η_i. Even if an adversary knew every other term from every other participant but 1, they could only construct n equations

relating $n+1$ unknowns. Thus, as there are more unknowns kept secret by the ith participant, no information is revealed.

Lastly to finish constructing the signature, we compute:

$$s \equiv \Gamma \mod N$$

And finalize the signature as:

$$\sigma = (r, s)$$

Now, we verify σ is a standard-conforming ECDSA against the public key B for message M. If this fails, our protocol has been compromised.

If the signature verifies, this serves as a consistency check that the signature was honestly computed and no key data was leaked.

The total number of Oblivious Transfer Multiplications required of this protocol is $n^2 - n$. However, each participant participates in only $2n - 2$ of them. The overall latency for producing a single signature is still bound by $n^2 - n$, but through depending on scheduling, n signatures may be produced in $2n^2 - 2n$ serial steps rather than $n^3 - n^2$.

With an application-specific optimization, where for each subsequent signature there are $n/2$ participants, this latency can be reduced to $4n$ serial steps. Thus running such a protocol with, say, 1000 participants and 256KiB OT Multiplications would require 1GB of bandwidth per participant. With 16KiB OT Multiplication Protocol, this would be reduced to 65MB, but would require more computation.

2.4 Batch Signing

If we run a single round of Reusable Nonce Generation, we may run P rounds of Single Use Key Generation and Single Signing. Provided all ECDSA verifications succeed, we have successfully signed P messages for P unique keys that are unforgeable by any $t < n$ party.

3 Use Cases

This signature scheme is secure for pre-signing a tree of transactions and then creating the root parent transaction. Because of the semi-honest model, it is not secure for non-presigned use cases.

All of these use cases work with either Schnorr signatures (with modification to Bitcoin), or in plaintext multisig (with cost of scalability and bound on number of participants).

3.1 Certified Post Dated UTXOs

A group of participants can construct a tree of presigned transactions that they are willing to accept as a payment from another set of participants.

The payees submit to the payers a request for an unsigned transaction which pays out the desired balance to an aggregated key.

The payer builds such a transaction, selecting their preferred inputs.

They payees, upon receiving this TXID, generate a binary tree of presigned transactions which pays out each participant at the leafs.

The payees, upon finishing the tree construction, instruct the payer to complete the payment.

The overall overhead of this protocol is $2 * n$ transactions, and for each individual $\log n$ transactions. For an participants redeeming randomly, the expected overhead is 2 transactions. Each branch in the CPDU has $n/2$ signers as it must only be trustless to those posessing a child CPDU.

This is sort of like a Certified cheque in that the balance is guaranteed, and is like a post-dated cheque in that the balance is only redeemable at some time later, hence the name. CPDU for short!

CPDU payments efficacy as a scaling solution is difficult to model, but in a maximal use case, there could be a single output per 'high volume' block, which could support 40 unique payers/per second paying to a practically unlimited number of recipients. In practice, this protocol is limited by the number of cooperating recipients and the need of recipients to eventually redeem.

3.2 UTXO Compression

An untrusting group of participants can perform the above CPDU protocol with all of their outstanding UTXOs.

Then, at a time later, only when needed, they can quickly ($O(\log N)$, $E[O(1)]$) re-create their desired UTXO.

This is similar to UTXO Commitment Proofs, but it uses on-chain bandwidth to expand the transactions.

3.3 CoinJoin

Using Mixnets to mask which participant selects which outputs, CPDU protocol can be used to make a more efficient CoinJoin which has delayed revelation of what the outputs are. Because of the key-aggregation, the branches of the coinjoin are indistinguishable from future payments.

3.4 Branching Protocols

If the CPDU protocol is done with multiple alternative branches (requiring a number of reusable nonces equal to the max branch width), CPDU can be used to develop complicated smart contracts which depend on other data or UTXOs which may be only potentially available.

A 1-of-N OT

Assuming we want to transfer 1-of-N r bit strings from Alice to Bob and only have 1-of-2 OT available. Alice select $\lceil 2 \log_2 N \rceil$ random r bit strings: $k^{b \in 0,1}_{j \in [0, 2\lceil \log_2 N \rceil)}$.

Alice encrypts each entry $m_i \in m_0, \ldots, m_{N-1}$ under all the keys $k_j^{b_n}$ such that the b_nth is the nth bit of i. E.g. for the $i = \texttt{0xAA}th$ entry out of 256,

$$enc(m_{\texttt{0xAA}}) = \epsilon_{k_0^1}(\epsilon_{k_1^0}(\epsilon_{k_2^1}(\epsilon_{k_3^0}(\epsilon_{k_4^1}(\epsilon_{k_5^0}(\epsilon_{k_6^1}(\epsilon_{k_7^0}(m_{\texttt{0xAA}}))))))))$$

Alice encrypts each entry as above and sends all the ciphertexts to Bob.

Alice then puts each keypair k_j^1, k_j^0 inside a 1-of-2 Oblivious Transfer and allows Bob to pick one.

Bob selects one out of two for each bit and after selecting $\lceil 2 \log_2 N \rceil$ keys, decrypts the result.

Any encryption scheme may be selected, but a XOR one-time-pad is sufficient.