Honey Badger Byzantine

Videos

https://youtu.be/kSqNfwIVfP4 - Lect-31-4010-pt1-news.mp4
https://youtu.be/4ZUhsp5CWa0 - Lect-31-4010-pt2-multi-sig.mp4
https://youtu.be/r3ui8hxSfdI - Lect-31-4010-pt3-homomorphic-Encryption.mp4

From Amazon S3 - for download (same as youtube videos)

http://uw-s20-2015.s3.amazonaws.com/Lect-31-4010-pt1-news.mp4
http://uw-s20-2015.s3.amazonaws.com/Lect-31-4010-pt2-multi-sig.mp4
http://uw-s20-2015.s3.amazonaws.com/Lect-31-4010-pt3-homomorphic-Encryption.mp4

News

1. China is officially testing a cryptocurrency as a national currency.
2. Australia is using a blockchain solution for registration of clinical trials.

Implementations

- go https://github.com/pschlump/hbbft my copy or Original https://github.com/anthdm/hbbft or anotehr implemenationGo https://githhub.com/poanetwork/hbbft
- Rust https://github.com/helium/erlang-hbbft
- Erlang https://github.com/helium/erlang-hbbft
- Python https://github.com/initc3/HoneyBadgerBFT-Python/

References

- The Honey Badger BFT protocols
- Practical Byzantine Fault Tolerance
- Treshold encryption
- Shared secret

Overview - How it works at a high level

Threshold ECDSA overview.

Details of t-ECDSA

Additively Homomorphic Encryption

- **Homomorphic addition of plaintexts**

  The product of two ciphertexts will decrypt to the sum of their corresponding plaintexts,

  $$D(E(m_1, r_1) \cdot E(m_2, r_2) \bmod n^2) = m_1 + m_2 \bmod n.$$

  The product of a ciphertext with a plaintext raising g will decrypt to the sum of the corresponding plaintexts,

  $$D(E(m_1, r_1) \cdot g^{m_2} \bmod n^2) = m_1 + m_2 \bmod n.$$

- **Homomorphic multiplication of plaintexts**

  An encrypted plaintext raised to the power of another plaintext will decrypt to the product of the two plaintexts,

  $$D(E(m_1, r_1)^{m_2} \bmod n^2) = m_1 m_2 \bmod n,$$
  $$D(E(m_2, r_2)^{m_1} \bmod n^2) = m_1 m_2 \bmod n.$$

  More generally, an encrypted plaintext raised to a constant *k* will decrypt to the product of the plaintext and the constant,

  $$D(E(m_1, r_1)^k \bmod n^2) = k m_1 \bmod n.$$

However, given the Paillier encryptions of two messages there is no known way to compute an encryption of the product of these messages without knowing the private key.

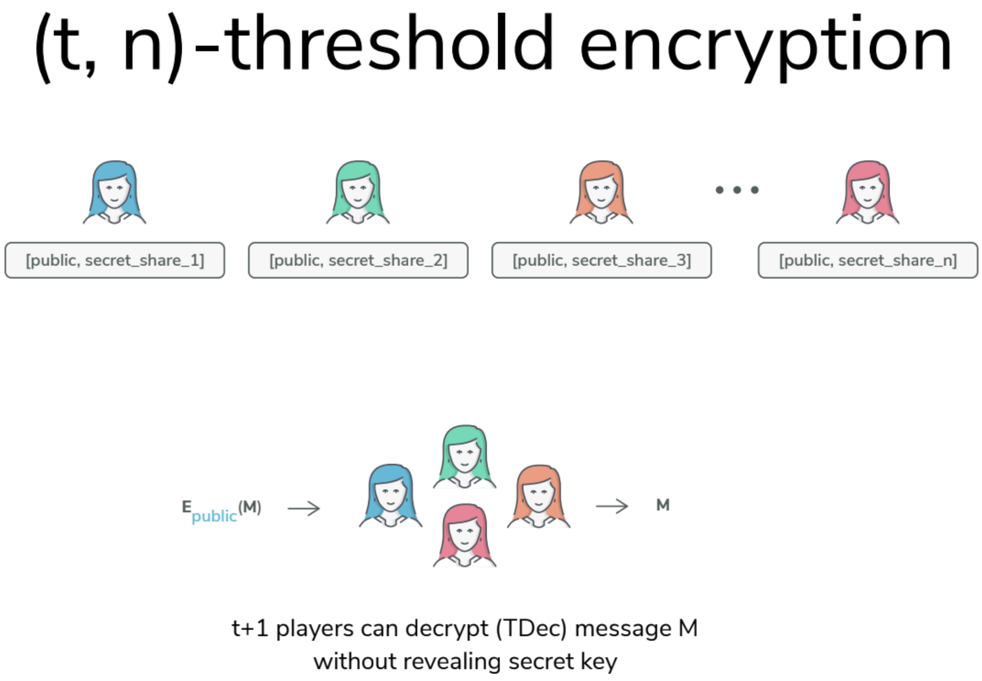
Additively homomorphic encryption has a nice feature, allowing to operate on ciphertexts. One example of additively homomorphic encryption scheme is Paillier, for which there exists an efficiently computable operation "add", so addition operation, that's plus with e subindex on the slide, that allows to add two ciphertexts together.

So, we can have two values, a and b. We can first encrypt those values and then, using this special operation, add those two ciphertexts together. Or we can first, having those two values, add them together and then encrypt the result. When we decrypt we will get exactly the same value in both cases.

Paillier Based Encryption - works like Elliptic Curve.

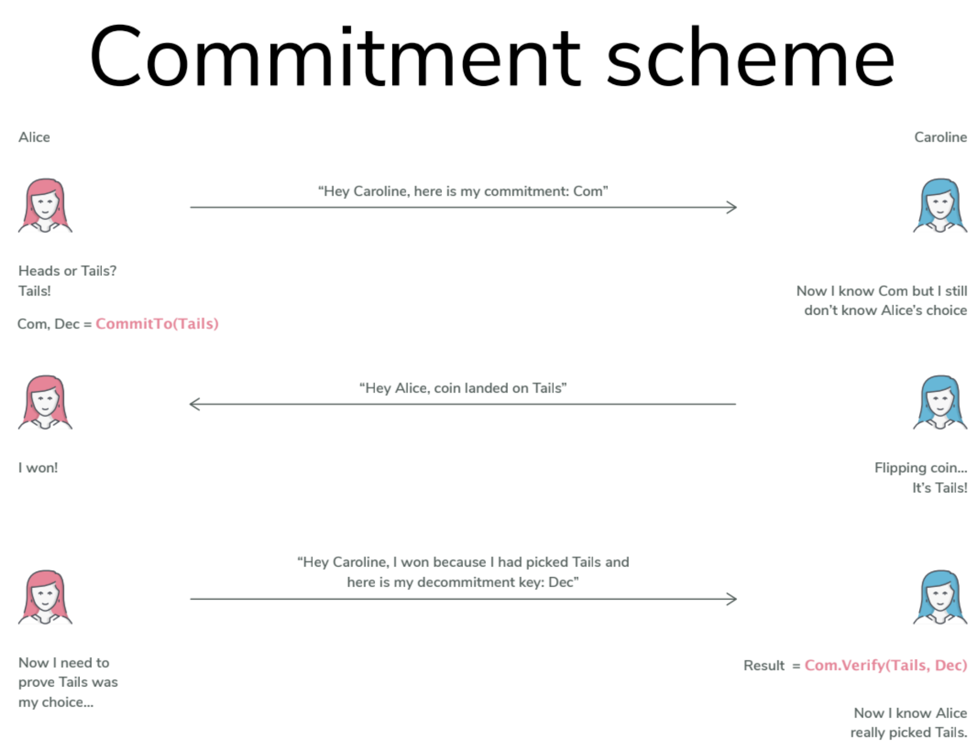
Threshold Encryption

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}$$



(t, n) threshold encryption. The easiest way to explain it is to say that we have n players; each one has the same public key, but also each player has its unique share of the secret key. So, on this slide we have secret_share 1, secret_share 2, secret_share 3…secret_share n, and signers—if you have a message encrypted under the public key, signers must cooperate using a special communication protocol that does not expose the secret key in order to decrypt it.

Commitment Scheme

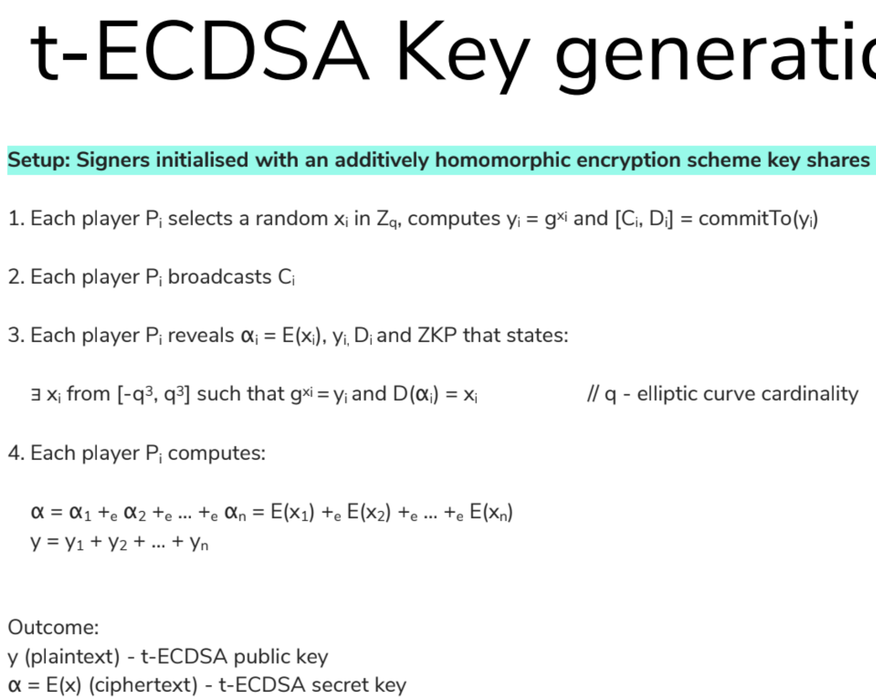


Let's say that Alice and Caroline are playing a coin flipping game. Alice chooses the side of the coin, evaluates a commitment to her choice, and sends the commitments to Caroline. Now Caroline throws the coin into the air and says what the result is. Next, Alice sends a special value called decommitment key, which allows Caroline to evaluate the commitment and to see if the value that was initially chosen by Alice is really what she's saying now.

The decommitment key allows to validate the commitment, but also has a nice feature, allowing to define unconditionally hiding schemes. So, no matter what Caroline does, no matter how high computing power she has, she cannot guess the value Alice committed to without having the decommitment key. She cannot do it from just the commitment.

Key Generation First

Assume we have n signers, each signer is initialized with the additively homomorphic threshold encryption scheme, and this happens in the setup phase:



This is from Threshold-optimal DSA/ECDSA signatures and
an application to Bitcoin wallet security
Rosario Gennaro, Steven Goldfeder, and Arvind Narayanan https://eprint.iacr.org/2016/013.pdf The paper is 42 pages long.

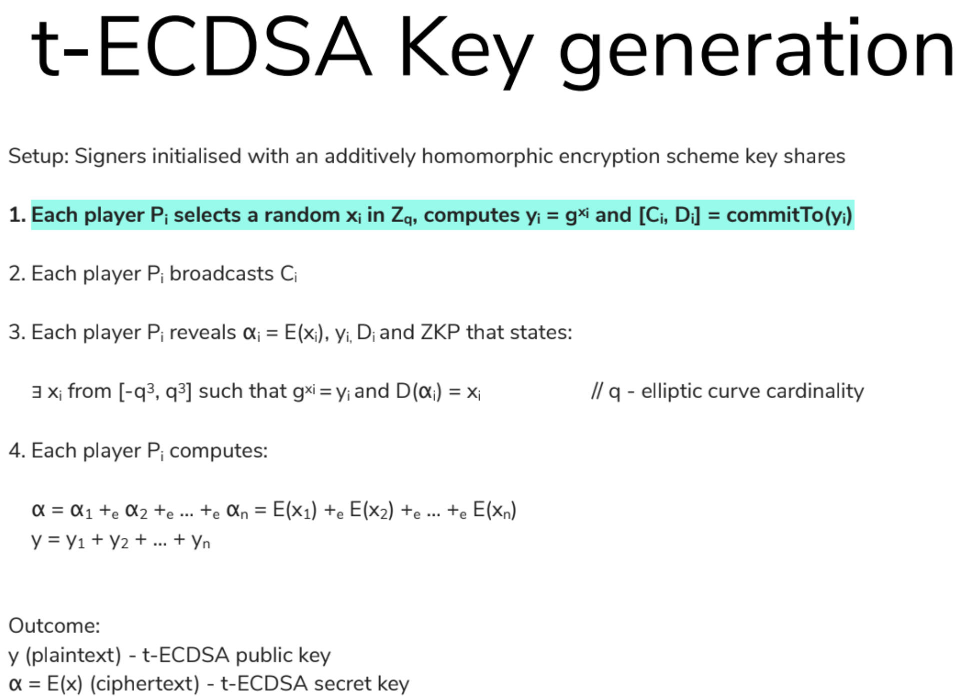
We have two types of keys. We have threshold additively homomorphic encryption scheme keys which are initialized in the set-up phase and we also have t-ECDSA key that we use for signing. And this second key is what we are going to generate now. For the additively homomorphic threshold encryption scheme, we just assume that it's done in the set-up phase.

Key Generation it's easy, because we can use a zero-knowledge proof and the commitnment to detect if one party misbehaves. If you have a bad party then replace them.

Use a trusted dealer that generates a set of keys, distributes them, then forgets them. Or execute some kind of a dealerless protocol in order to execute the keys by all group members together. See https://link.springer.com/chapter/10.1007/978-3-642-17955-6_4

Key Generation : Choose a random 'x'

In the first step, each player chooses a random integer x, which will be used as a secret key share of that player:



`x` cannot be greater than `q`. `q` is the elliptic curve cardinality, so it's the number of points elliptic curve has. On all of the slides, `q` stands for the cardinality of the elliptic curve. Each player computes `y` as `g` to the power of `x`. This is an elliptic curve operation, and basically, we multiply curve's generator point by `x`. This is a notation you will often find for groups, but since elliptic curve is a group we can use it also here.

Key Generation : Generate Commitments and Broadcast

Then, each player computes a commitment to this value, and in the second round, publishes this commitment to all of the players in the group:

t-ECDSA Key generation

Setup: Signers initialised with an additively homomorphic encryption scheme key shares

1. Each player P_i selects a random x_i in Z_q, computes $y_i = g^{x_i}$ and $[C_i, D_i] = \text{commitTo}(y_i)$

2. **Each player P_i broadcasts C_i**

3. Each player P_i reveals $\alpha_i = E(x_i)$, y_i, D_i and ZKP that states:

   $\exists\, x_i$ from $[-q^3, q^3]$ such that $g^{x_i} = y_i$ and $D(\alpha_i) = x_i$ // q - elliptic curve cardinality

4. Each player P_i computes:

   $\alpha = \alpha_1 +_e \alpha_2 +_e \dots +_e \alpha_n = E(x_1) +_e E(x_2) +_e \dots +_e E(x_n)$
   $y = y_1 + y_2 + \dots + y_n$

Outcome:
y (plaintext) - t-ECDSA public key
$\alpha = E(x)$ (ciphertext) - t-ECDSA secret key

Key Generation: Reveal encrypted ‘x’

Each player reveals `x` in an encrypted form, it’s `α` (alpha) below:

t-ECDSA Key generation

Setup: Signers initialised with an additively homomorphic encryption scheme key shares

1. Each player P_i selects a random x_i in Z_q, computes $y_i = g^{x_i}$ and $[C_i, D_i] = \text{commitTo}(y_i)$

2. Each player P_i broadcasts C_i

3. **Each player P_i reveals $\alpha_i = E(x_i)$, y_i, D_i and ZKP that states:**

   $\exists\, x_i$ from $[-q^3, q^3]$ such that $g^{x_i} = y_i$ and $D(\alpha_i) = x_i$ // q - elliptic curve cardinality

4. Each player P_i computes:

   $\alpha = \alpha_1 +_e \alpha_2 +_e \dots +_e \alpha_n = E(x_1) +_e E(x_2) +_e \dots +_e E(x_n)$
   $y = y_1 + y_2 + \dots + y_n$

Outcome:
y (plaintext) - t-ECDSA public key
$\alpha = E(x)$ (ciphertext) - t-ECDSA secret key

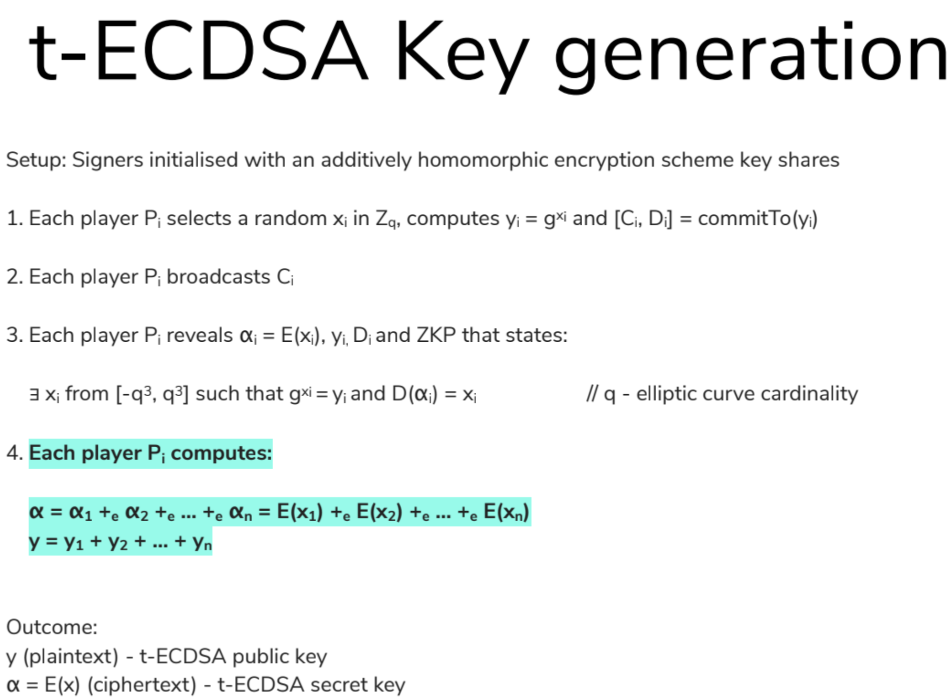
Encryption is done with the additively homomorphic encryption scheme we initialized in the set-up phase. What’s more, each player reveals the public key share, the decommitment key, and the zero-knowledge proof stating that all those values together make sense. This allows for a validation of the commitment, but also allows all players to see if all those shares that the player just revealed together makes some sense.

So, the zero-knowledge proof says that there exists a number `x` such that curve's generator point multiplied by `x` gives point `y` , and `y` is public at this moment because it was just revealed, and that if we decrypt the value α that we have just published—it's the encrypted secret key—we'll get that number `x` .

Of course, it's a zero-knowledge proof, so it is not possible to guess what `x` really is, but what we say here is that it lies in the range from $(-q^3, q^3)$, and since `q` is the cardinality of the elliptic curve, this range is really huge.

Key Generation: Generate ECDSA final key

All signers use the add operation of the additively homomorphic threshold scheme to produce the final t-ECDSA key, so all of the encrypted shares of x can be added:



We get the secret key in an encrypted form, all revealed public shares of y can be added, and as a result, we have the public key. Addition operation here is just addition of elliptic curve points, so it's easy.

Signature

Signature: Step 1 and 2

We have `α` , which is a t-ECDSA private key in encrypted form shared between all the signers, and we have `y` which is a t-ECDSA public key. Public keys are just a point on the elliptic curve.

t-ECDSA Signature

Setup: Signers initialised with t-ECDSA key [α, y];
α = E(x) - encrypted private key
y - public key

In the first round each party draws a random integer ρ :

t-ECDSA Signature

Round 1 and 2

Using the commit and reveal pattern, each player P_i:

- selects a random ρ_i in Z_q
- computes $u_i = E(\rho_i)$
- computes $v_i = \rho_i \times_e \alpha = E(\rho_i x)$
- provides ZKP which states u_i and v_i are correct

After the round 2, all players join shares together:

- $u = \sum u_i = E(\rho)$
- $v = \sum v_i = E(\rho x)$

Encrypt the value with additively homomorphic threshold encryption scheme:

t-ECDSA Signature

Round 1 and 2

Using the commit and reveal pattern, each player P_i:

- selects a random ρ_i in Z_q
- computes $u_i = E(\rho_i)$
- computes $v_i = \rho_i \times_e \alpha = E(\rho_i x)$
- provides ZKP which states u_i and v_i are correct

After the round 2, all players join shares together:

- $u = \sum u_i = E(\rho)$
- $v = \sum v_i = E(\rho x)$

Multiply the secret ECDSA key (α below) by this random value:

t-ECDSA Signature

Round 1 and 2

Using the commit and reveal pattern, each player P_i:

- selects a random ρ_i in Z_q
- computes $u_i = E(\rho_i)$
- computes $v_i = \rho_i \times_e \alpha = E(\rho_i x)$
- provides ZKP which states u_i and v_i are correct

After the round 2, all players join shares together:

- $u = \sum u_i = E(\rho)$
- $v = \sum v_i = E(\rho x)$

Use addition to implement multiplication. Each signer publishes commitment to those values, and in the second round reveals all those values, along with the zero-knowledge proof, stating that they make sense:

t-ECDSA Signature

Round 1 and 2

Using the commit and reveal pattern, each player P_i:

- selects a random ρ_i in Z_q
- computes $u_i = E(\rho_i)$
- computes $v_i = \rho_i \times_e \alpha = E(\rho_i x)$
- provides ZKP which states u_i and v_i are correct

After the round 2, all players join shares together:

- $u = \sum u_i = E(\rho)$
- $v = \sum v_i = E(\rho x)$

Using zero-knowledge proof and the commitment we can now prove that the values we have are correct.

Join all the shares together (This means broadcast all the shares so that everybody has them):

Reveal the commitments:

t-ECDSA Signature

Round 3 and 4

Using the commit and reveal pattern, each player P_i:

- selects a random k_i in Z_q
- selects a random c_i from $[-q^6, q^6]$
- computes $r_i = g^{k_i}$
- computes $w_i = E(k_i \rho + c_i q)$
- provides ZKP which states r_i and w_i are correct

After the round 4, all players join shares together:

- $k = \sum k_i$
- $c = \sum c_i$
- $w = \sum w_i = E(k\rho + cq)$
- $r = H(\sum r_i) = H(g^k)$

After the round 2, all players know those values:

- $u = \sum u_i = E(\rho)$ $\quad u_i \in Z_q$
- $v = \sum v_i = E(\rho x)$

Signature: Step 3 and 4

The same commit-reveal pattern is used in the 3rd and 4th round:

t-ECDSA Signature

Round 3 and 4

Using the commit and reveal pattern, each player P_i:

- selects a random k_i in Z_q
- selects a random c_i from $[-q^6, q^6]$
- computes $r_i = g^{k_i}$
- computes $w_i = E(k_i \rho + c_i q)$
- provides ZKP which states r_i and w_i are correct

After the round 4, all players join shares together:

- $k = \sum k_i$
- $c = \sum c_i$
- $w = \sum w_i = E(k\rho + cq)$
- $r = H(\sum r_i) = H(g^k)$

After the round 2, all players know those values:

- $u = \sum u_i = E(\rho)$ $u_i \in Z_q$
- $v = \sum v_i = E(\rho x)$

On the right side we have all the parameters that were evaluated so far, and all players have the same values.

In the third round each party draws a random integer `ki`:

t-ECDSA Signature

Round 3 and 4

Using the commit and reveal pattern, each player P_i:

- selects a random k_i in Z_q
- selects a random c_i from $[-q^6, q^6]$
- computes $r_i = g^{k_i}$
- computes $w_i = E(k_i \rho + c_i q)$
- provides ZKP which states r_i and w_i are correct

After the round 4, all players join shares together:

- $k = \sum k_i$
- $c = \sum c_i$
- $w = \sum w_i = E(k\rho + cq)$
- $r = H(\sum r_i) = H(g^k)$

After the round 2, all players know those values:

- $u = \sum u_i = E(\rho)$ $u_i \in Z_q$
- $v = \sum v_i = E(\rho x)$

And a random integer `ci`:

t-ECDSA Signature

Round 3 and 4

Using the commit and reveal pattern, each player P_i:

- selects a random k_i in Z_q
- selects a random c_i from $[-q^6, q^6]$
- computes $r_i = g^{k_i}$
- computes $w_i = E(k_i \rho + c_i q)$
- provides ZKP which states r_i and w_i are correct

After the round 4, all players join shares together:

- $k = \sum k_i$
- $c = \sum c_i$
- $w = \sum w_i = E(k\rho + cq)$
- $r = H(\sum r_i) = H(g^k)$

After the round 2, all players know those values:

- $u = \sum u_i = E(\rho)$ $u_i \in Z_q$
- $v = \sum v_i = E(\rho x)$

`q` all the time stands for the cardinality of the elliptic curve, so the number of points on the elliptic curve.

Each party computes `rᵢ` as `g` to the power of `k` —we basically multiply curve generator point by it:

t-ECDSA Signature

Round 3 and 4

Using the commit and reveal pattern, each player P_i:

- selects a random k_i in Z_q
- selects a random c_i from $[-q^6, q^6]$
- computes $r_i = g^{k_i}$
- computes $w_i = E(k_i \rho + c_i q)$
- provides ZKP which states r_i and w_i are correct

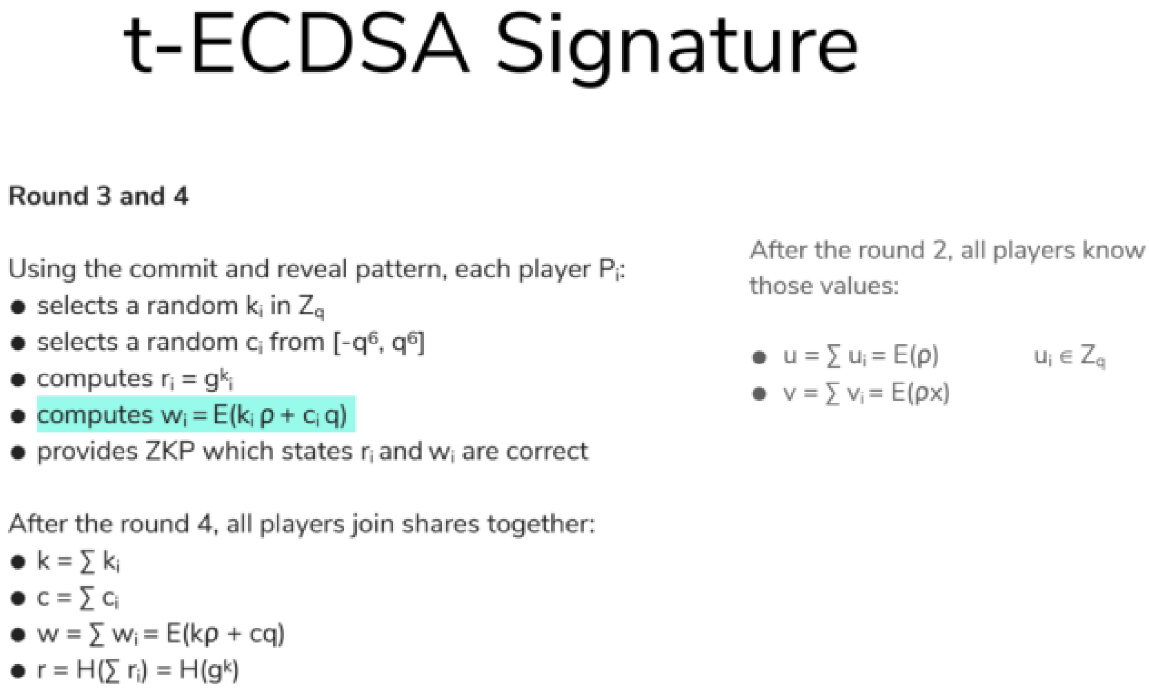
After the round 4, all players join shares together:

- $k = \sum k_i$
- $c = \sum c_i$
- $w = \sum w_i = E(k\rho + cq)$
- $r = H(\sum r_i) = H(g^k)$

After the round 2, all players know those values:

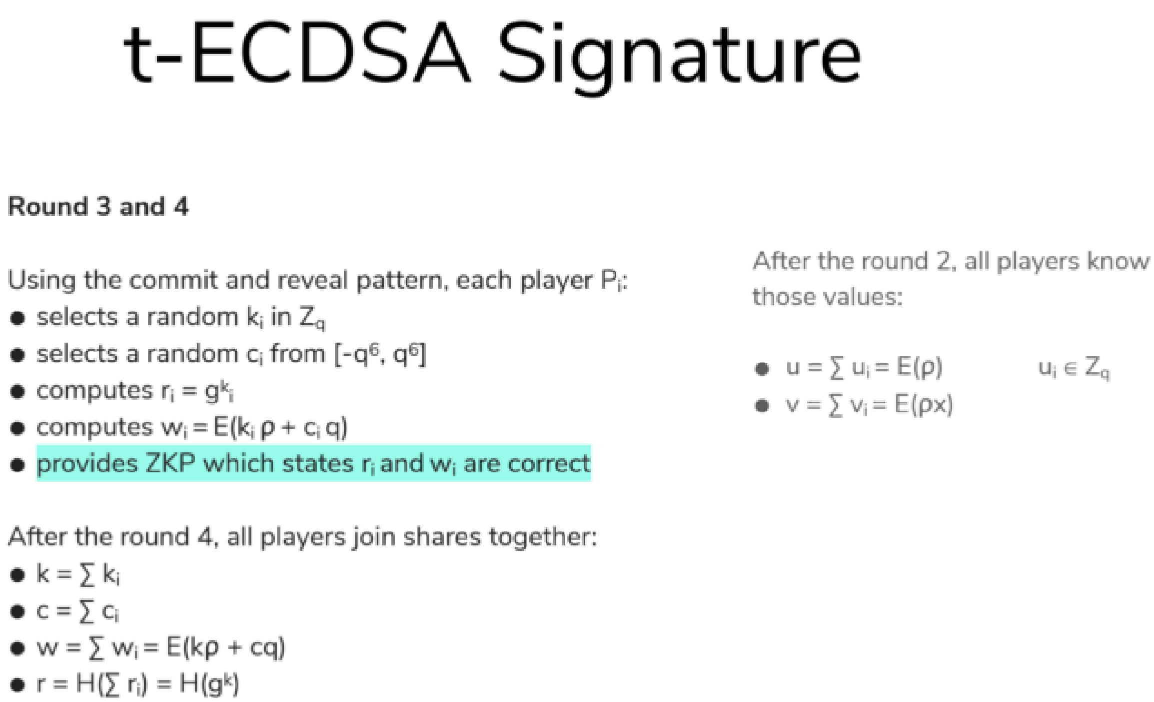
- $u = \sum u_i = E(\rho)$ $u_i \in Z_q$
- $v = \sum v_i = E(\rho x)$

Each party computes the parameter `w` which is `k` time `ρ` plus `c` times `q`:



`q` all the time is the cardinality of the elliptic curve, and we can compute it because we use additively homomorphic threshold encryption.

At the end, each party commits to all those parameters, and in the round 4 generated parameters are revealed, along with the zero-knowledge proof stating that they make sense together:



Having all those parameters from all the group members we can add them together, just like we did after the round 2. We sum up all `k` shares, all `c` shares, all `w` shares. We evaluate parameter `r` as a sum of all `ri` shares and we use a special hash function:

t-ECDSA Signature

Round 3 and 4

Using the commit and reveal pattern, each player P_i:

- selects a random k_i in Z_q
- selects a random c_i from $[-q^6, q^6]$
- computes $r_i = g^{k_i}$
- computes $w_i = E(k_i \rho + c_i q)$
- provides ZKP which states r_i and w_i are correct

After the round 4, all players join shares together:

- $k = \sum k_i$
- $c = \sum c_i$
- $w = \sum w_i = E(k\rho + cq)$
- $r = H(\sum r_i) = H(g^k)$

After the round 2, all players know those values:

- $u = \sum u_i = E(\rho)$ $u_i \in Z_q$
- $v = \sum v_i = E(\rho x)$

This is the standard ECDSA. `x` is a point coordinate modulo the `q` elliptic curve order.

Signature: Step 5

All parameters on the right side are shared by all signers in the group.

Now we need to do some discrete mathematics magic to produce a signature. Using all those parameters we have evaluated so far, and since we operate on encrypted data, the signature will be also encrypted. But this is something we will deal in the final round 6.

t-ECDSA Signature

Round 5

All players jointly decrypt w:
$\eta = TDec(w) = k\rho \bmod q$

and compute:
$\Psi = \eta^{-1} \bmod q = k^{-1} \rho^{-1}$

$$\begin{aligned}\sigma &= \Psi \times_e [(m \times_e u) +_e (r \times_e v)] \\ &= \Psi \times_e [E(m\rho) +_e E(r\rho x)] \\ &= (k^{-1} \rho^{-1}) \times_e [E(\rho(m + xr))] \\ &= E(k^{-1} (m + xr)) \\ &= E(s)\end{aligned}$$

After round 4, all players know those values:

- $u = \sum u_i = E(\rho)$ $u_i \in Z_q$
- $v = \sum v_i = E(\rho x)$
- $k = \sum k_i$ $k_i \in Z_q$
- $c = \sum c_i$ $c_i \in [-q^6, q^6]$
- $w = \sum w_i = E(k\rho + cq)$
- $r = H(\sum r_i) = H(g^k)$

The very first thing we need to do is that we execute a threshold decryption mechanism to have all the players decrypt the parameter `w` and assign this value to `η`:

t-ECDSA Signature

Round 5

All players jointly decrypt w:
$\eta = TDec(w) = k\rho \bmod q$

and compute:
$\Psi = \eta^{-1} \bmod q = k^{-1}\rho^{-1}$

$$\begin{aligned}\sigma &= \Psi \times_e [(m \times_e u) +_e (r \times_e v)]\\ &= \Psi \times_e [E(m\rho) +_e E(r\rho x)]\\ &= (k^{-1}\rho^{-1}) \times_e [E(\rho(m + xr))]\\ &= E(k^{-1}(m + xr))\\ &= E(s)\end{aligned}$$

After round 4, all players know those values:

- $u = \sum u_i = E(\rho)$ $\quad u_i \in Z_q$
- $v = \sum v_i = E(\rho x)$
- $k = \sum k_i$ $\quad k_i \in Z_q$
- $c = \sum c_i$ $\quad c_i \in [-q^6, q^6]$
- $w = \sum w_i = E(k\rho + cq)$
- $r = H(\sum r_i) = H(g^k)$

Compute yet one parameter called `Ψ` which is multiplicative inverse of `η` modulo `q`, and `q` is all the time cardinality of the elliptic curve:

t-ECDSA Signature

Round 5

All players jointly decrypt w:
$\eta = TDec(w) = k\rho \bmod q$

and compute:
$\Psi = \eta^{-1} \bmod q = k^{-1}\rho^{-1}$

$$\begin{aligned}\sigma &= \Psi \times_e [(m \times_e u) +_e (r \times_e v)]\\ &= \Psi \times_e [E(m\rho) +_e E(r\rho x)]\\ &= (k^{-1}\rho^{-1}) \times_e [E(\rho(m + xr))]\\ &= E(k^{-1}(m + xr))\\ &= E(s)\end{aligned}$$

After round 4, all players know those values:

- $u = \sum u_i = E(\rho)$ $\quad u_i \in Z_q$
- $v = \sum v_i = E(\rho x)$
- $k = \sum k_i$ $\quad k_i \in Z_q$
- $c = \sum c_i$ $\quad c_i \in [-q^6, q^6]$
- $w = \sum w_i = E(k\rho + cq)$
- $r = H(\sum r_i) = H(g^k)$

Having `m`, the hash of the message we are signing (or a hash of the transaction), we start evaluating the signature with the following equation:

t-ECDSA Signature

Round 5

All players jointly decrypt w:
$\eta = TDec(w) = k\rho \bmod q$

and compute:
$\Psi = \eta^{-1} \bmod q = k^{-1}\rho^{-1}$

$$\begin{aligned}\sigma &= \Psi \times_e [(m \times_e u) +_e (r \times_e v)]\\ &= \Psi \times_e [E(m\rho) +_e E(r\rho x)]\\ &= (k^{-1}\rho^{-1}) \times_e [E(\rho(m + xr))]\\ &= E(k^{-1}(m + xr))\\ &= E(s)\end{aligned}$$

After round 4, all players know those values:

- $u = \sum u_i = E(\rho)$ $u_i \in Z_q$
- $v = \sum v_i = E(\rho x)$
- $k = \sum k_i$ $k_i \in Z_q$
- $c = \sum c_i$ $c_i \in [-q^6, q^6]$
- $w = \sum w_i = E(k\rho + cq)$
- $r = H(\sum r_i) = H(g^k)$

`c` is the value we have just evaluated, and `u`, `r` and `v` are the parameters jointly evaluated by all the signers in previous rounds.

So, since `u` is an encrypted `ρ`, and `v` is an encrypted `ρ` multiplied by the secret ECDSA key, we can do the following transformation:

t-ECDSA Signature

Round 5

All players jointly decrypt w:
$\eta = TDec(w) = k\rho \bmod q$

and compute:
$\Psi = \eta^{-1} \bmod q = k^{-1}\rho^{-1}$

$$\begin{aligned}\sigma &= \Psi \times_e [(m \times_e u) +_e (r \times_e v)]\\ &= \Psi \times_e [E(m\rho) +_e E(r\rho x)]\\ &= (k^{-1}\rho^{-1}) \times_e [E(\rho(m + xr))]\\ &= E(k^{-1}(m + xr))\\ &= E(s)\end{aligned}$$

After round 4, all players know those values:

- $u = \sum u_i = E(\rho)$ $u_i \in Z_q$
- $v = \sum v_i = E(\rho x)$
- $k = \sum k_i$ $k_i \in Z_q$
- $c = \sum c_i$ $c_i \in [-q^6, q^6]$
- $w = \sum w_i = E(k\rho + cq)$
- $r = H(\sum r_i) = H(g^k)$

Replace `Ψ` with the value it represents, we will get the following equation:

t-ECDSA Signature

Round 5

All players jointly decrypt w:
$\eta = TDec(w) = k\rho \bmod q$

and compute:
$\Psi = \eta^{-1} \bmod q = k^{-1} \rho^{-1}$

$\sigma = \Psi \times_e [(m \times_e u) +_e (r \times_e v)]$
$= \Psi \times_e [E(m\rho) +_e E(r\rho x)]$
$= (k^{-1} \rho^{-1}) \times_e [E(\rho(m + xr))]$
$= E(k^{-1} (m + xr))$
$= E(s)$

After round 4, all players know those values:

- $u = \sum u_i = E(\rho)$ $u_i \in Z_q$
- $v = \sum v_i = E(\rho x)$
- $k = \sum k_i$ $k_i \in Z_q$
- $c = \sum c_i$ $c_i \in [-q^6, q^6]$
- $w = \sum w_i = E(k\rho + cq)$
- $r = H(\sum r_i) = H(g^k)$

Eliminate ρ, we get this:

t-ECDSA Signature

Round 5

All players jointly decrypt w:
$\eta = TDec(w) = k\rho \bmod q$

and compute:
$\Psi = \eta^{-1} \bmod q = k^{-1} \rho^{-1}$

$\sigma = \Psi \times_e [(m \times_e u) +_e (r \times_e v)]$
$= \Psi \times_e [E(m\rho) +_e E(r\rho x)]$
$= (k^{-1} \rho^{-1}) \times_e [E(\rho(m + xr))]$
$= E(k^{-1} (m + xr))$
$= E(s)$

After round 4, all players know those values:

- $u = \sum u_i = E(\rho)$ $u_i \in Z_q$
- $v = \sum v_i = E(\rho x)$
- $k = \sum k_i$ $k_i \in Z_q$
- $c = \sum c_i$ $c_i \in [-q^6, q^6]$
- $w = \sum w_i = E(k\rho + cq)$
- $r = H(\sum r_i) = H(g^k)$

This is the equation is forthe standard ECDSA signature, where `k` is the cryptographically secure random integer, `n` is the message hash, `x` is our secret ECDSA key, and `r` is the curve generated point multiplied `k` times modulo `q`.

All those equations were done on ciphertexts, so at the end our signature is also encrypted:

t-ECDSA Signature

Round 5

All players jointly decrypt w:
$\eta = TDec(w) = k\rho \bmod q$

and compute:
$\Psi = \eta^{-1} \bmod q = k^{-1}\ \rho^{-1}$

$$\begin{aligned}\sigma &= \Psi \times_e [(m \times_e u) +_e (r \times_e v)] \\ &= \Psi \times_e [E(m\rho) +_e E(r\rho x)] \\ &= (k^{-1}\ \rho^{-1}) \times_e [E(\rho(m + xr))] \\ &= E(k^{-1}(m + xr)) \\ &= E(s)\end{aligned}$$

After round 4, all players know those values:

- $u = \sum u_i = E(\rho)$ $\quad u_i \in Z_q$
- $v = \sum v_i = E(\rho x)$
- $k = \sum k_i$ $\quad k_i \in Z_q$
- $c = \sum c_i$ $\quad c_i \in [-q^6, q^6]$
- $w = \sum w_i = E(k\rho + cq)$
- $r = H(\sum r_i) = H(g^k)$

Signature: Step 6 - Deal with encrypted results

All the players execute a threshold decryption mechanism to learn the value of `s`. And the decrypted value `s` and parameter `r` evaluated in round 4 together make the signature:

t-ECDSA Signature

Round 6

Players execute distributed decryption protocol TDec over the ciphertext $\sigma = E(s)$ to learn the value of s.

Players outputs (r, s) as the signature.