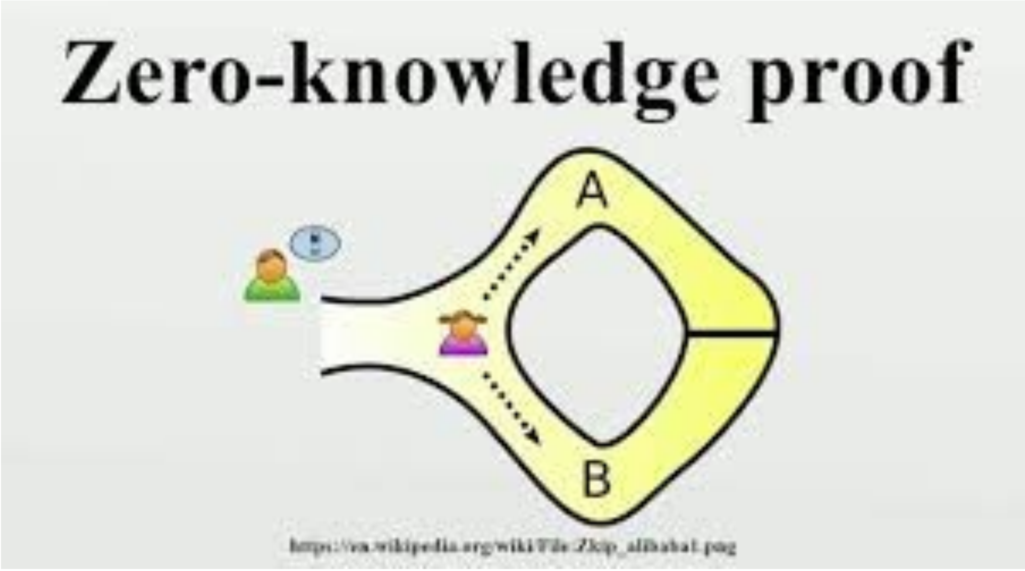
# ZKP and Block Chain

2019.10.24 (목) 한규형

# Zero Knowledge Proof (ZKP)

**Cave problem**



# Zero Knowledge Proof (ZKP)

## Schnorr protocol

**Protocol $\Pi_{\mathsf{dlog}}$**

**Common Input:** the description of a prime-order group $\mathbb{G}$ of (exponentially large) order $p$ with a generator $g$, and a group element $h$.

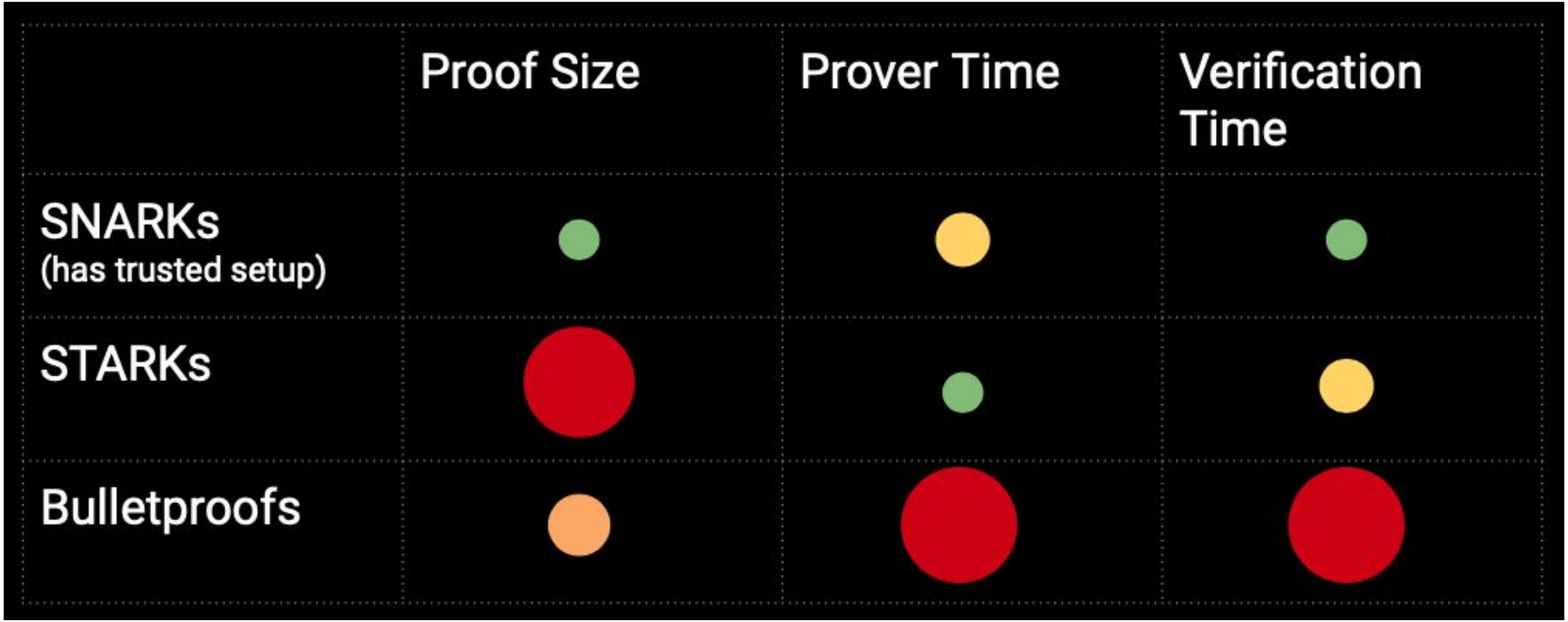
**Prover Witness:** A value $x \in \mathbb{Z}_p$ such that $g^x = h$.

**Protocol:**

1. $\mathcal{P}$: pick $r \xleftarrow{\$} \mathbb{Z}_p$, send $\rho \leftarrow g^r$.
2. $\mathcal{V}$: pick $e \xleftarrow{\$} \mathbb{Z}_p$, send $e$.
3. $\mathcal{P}$: send $d \leftarrow e \cdot x + r \bmod p$

**Verification:** $\mathcal{V}$ accepts iff $g^d = h^e \rho$.

# Zero Knowledge Proof (ZKP)



# ZKP - Sumcheck

**Definition 4** *The **Sumcheck problem** is the problem of proving that evaluations of the arithmetization of a boolean formula over the Boolean hypercube sum up to a value $s$. All arithmetic is performed in a finite field $\mathbb{Z}_q$ that is large enough to represent the result of the summation.*

The **Sumcheck protocol** (see Figure 1) solves the **Sumcheck problem**. Informally, the protocol proceeds as follows:

1. The prover $\mathcal{P}$ and verifier $\mathcal{V}$ are given as input a boolean formula $\varphi$ and a field element $s$. Both arithmetize $\varphi$ to obtain a polynomial $p$ in $n$ variables $x_1, \ldots, x_n$.
2. $\mathcal{V}$ generates a random prime $q$ that is greater than $2^n 3^m$ and sends it to $\mathcal{P}$.
3. $\mathcal{V}$ initializes the $0^{th}$ *check-value* $v_0 := s$.
4. The following interaction is repeated for all $i = 1$ to $n$:
   (a) Leaving $x_i$ free, $\mathcal{P}$ evaluates $p$ at $x_{i+1} \in \{0,1\}, \ldots, x_n \in \{0,1\}$ to obtain polynomial $p_i$ in $x_i$:
   $$p_i(x_i) := \sum_{x_{i+1},\ldots,x_n \in \{0,1\}} p(r_1, r_2, \ldots, x_i, \ldots, x_n) \quad .$$
   $\mathcal{P}$ sends $p_i$ over to $\mathcal{V}$.
   (b) $\mathcal{V}$ checks that $p_i(0) + p_i(1) = v_{i-1}$. If so, it samples a random field element $r_i$, computes the next check-value $v_i := p_i(r_i)$, and sends $r_i$ to $\mathcal{P}$.
5. In the final round, instead of sending $r_n$ over to $\mathcal{P}$, $\mathcal{V}$ checks that $p(r_1, \ldots, r_n) = v_n$.

# ZKP - Sumcheck

**Definition 4** *The **Sumcheck problem** is the problem of proving that evaluations of the arithmetization of a boolean formula over the Boolean hypercube sum up to a value $s$. All arithmetic is performed in a finite field $\mathbb{Z}_q$ that is large enough to represent the result of the summation.*

The **Sumcheck protocol** (see Figure 1) solves the **Sumcheck problem**. Informally, the protocol proceeds as follows:

1. The prover $\mathcal{P}$ and verifier $\mathcal{V}$ are given as input a boolean formula $\varphi$ and a field element $s$. Both arithmetize $\varphi$ to obtain a polynomial $p$ in $n$ variables $x_1, \ldots, x_n$.
2. $\mathcal{V}$ generates a random prime $q$ that is greater than $2^n 3^m$ and sends it to $\mathcal{P}$.
3. $\mathcal{V}$ initializes the check-value $v_0 := s$.
4. The following interaction is repeated for all $i = 1$ to $n$:
   (a) Leaving $x_i$ free, $\mathcal{P}$ evaluates $p$ at $x_{i+1} \in \{0,1\}, \ldots, x_n \in \{0,1\}$ to obtain polynomial $p_i$ in $x_i$:
   $$p_i(x_i) := \sum_{x_{i+1},\ldots,x_n \in \{0,1\}} p(r_1, r_2, \ldots, x_i, \ldots, x_n) \quad .$$
   $\mathcal{P}$ sends $p_i$ over to $\mathcal{V}$.
   (b) $\mathcal{V}$ checks that $p_i(0) + p_i(1) = v_{i-1}$. If so, it samples a random field element $r_i$, computes the next check-value $v_i := p_i(r_i)$, and sends $r_i$ to $\mathcal{P}$.
5. In the final round, instead of sending $r_n$ over to $\mathcal{P}$, $\mathcal{V}$ checks that $p(r_1, \ldots, r_n) = v_n$.

- Compute needs 2^n computations
- Verification only needs n computations

# ZKP - Verifiable Computation (VC)

GKR protocol - MLE, Sum-check (Libra, Hyrax = ZK version of GKR)

$$\tilde{W}_i(\mathbf{z}) = \sum_{\mathbf{b},\mathbf{c}\in\{0,1\}^{s_{i+1}}} \underline{\widetilde{\mathrm{add}}_i(\mathbf{z},\mathbf{b},\mathbf{c})}\left(\tilde{W}_i(\mathbf{b}) + \tilde{W}_i(\mathbf{c})\right) + \underline{\widetilde{\mathrm{mult}}_i(\mathbf{z},\mathbf{b},\mathbf{c})}\left(\tilde{W}_i(\mathbf{b}) \cdot \tilde{W}_i(\mathbf{c})\right)$$

- **Add_i(x, y, z)** = 1 if (y, z) is input of add gate x in layer i
- **Mult_i(x, y, z)** = 1 if (y, z) is input of mult gate x in layer i
- **V_i(x)** = output of gate x in layer i

# ZKP - Verifiable Computation (VC)

GKR protocol - MLE, Sum-check (Libra, Hyrax = ZK version of GKR)

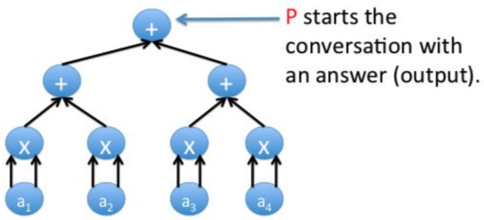



Figure 1: Start of GKR Protocol.

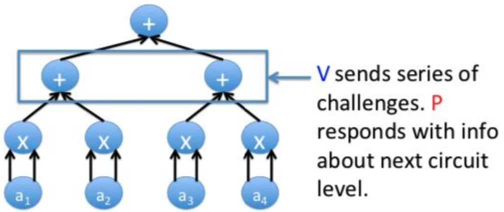



Figure 2: Iteration 1 reduces a claim about the output of $\mathcal{C}$ to one about the MLE of the gate values in the previous layer.

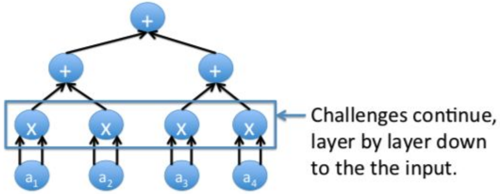



Figure 3: In general, iteration $i$ reduces a claim about the MLE of gate values at layer $i$, to a claim about the MLE of gate values at layer $i+1$.

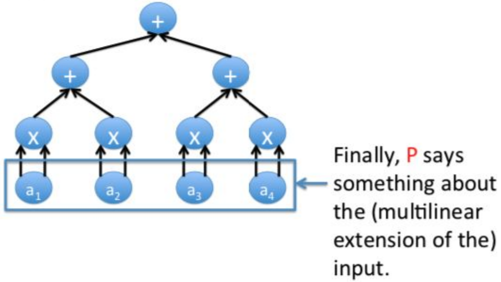
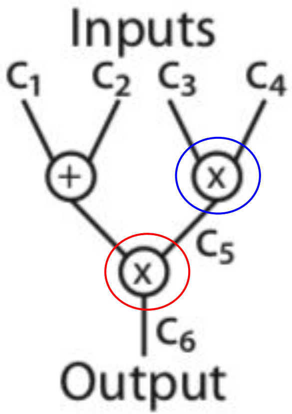



Figure 4: In the final iteration, $\mathcal{P}$ makes a claim about the MLE of the input. $\mathcal{V}$ can check this claim without help, since $\mathcal{V}$ sees the input explicitly.

# ZKP - Verifiable Computation (VC)

Pinocchio - QAP, QSP




| | $(r_5, r_6)$ | | $(r_5, r_6)$ | | $(r_5, r_6)$ |
|---|---|---|---|---|---|
| $v_1(r_i)$ | (0,1) | $w_1(r_i)$ | (0,0) | $y_1(r_i)$ | (0,0) |
| $v_2(r_i)$ | (0,1) | $w_2(r_i)$ | (0,0) | $y_2(r_i)$ | (0,0) |
| $v_3(r_i)$ | (1,0) | $w_3(r_i)$ | (0,0) | $y_3(r_i)$ | (0,0) |
| $v_4(r_i)$ | (0,0) | $w_4(r_i)$ | (1,0) | $y_4(r_i)$ | (0,0) |
| $v_5(r_i)$ | (0,0) | $w_5(r_i)$ | (0,1) | $y_5(r_i)$ | (1,0) |
| $v_6(r_i)$ | (0,0) | $w_6(r_i)$ | (0,0) | $y_6(r_i)$ | (0,1) |

$$t(x) = (x - r_5)(x - r_6)$$

Figure 2: **Arithmetic Circuit and Equivalent QAP.** *Each wire value comes from, and all operations are performed over, a field $\mathbb{F}$. The polynomials in the QAP are defined in terms of their evaluations at the two roots, $r_5$ and $r_6$. See text for details.*

# ZKP - Verifiable Computation (VC)

## Pinocchio - QAP, QSP

**Definition 2 (Quadratic Arithmetic Program (QAP) [30])** *A QAP $Q$ over field $\mathbb{F}$ contains three sets of $m+1$ polynomials $\mathcal{V} = \{v_k(x)\}$, $\mathcal{W} = \{w_k(x)\}$, $\mathcal{Y} = \{y_k(x)\}$, for $k \in \{0 \ldots m\}$, and a target polynomial $t(x)$. Suppose $F$ is a function that takes as input $n$ elements of $\mathbb{F}$ and outputs $n'$ elements, for a total of $N = n + n'$ I/O elements. Then we say that $Q$ computes $F$ if: $(c_1, \ldots, c_N) \in \mathbb{F}^N$ is a valid assignment of $F$'s inputs and outputs,* if and only if *there exist coefficients $(c_{N+1}, \ldots, c_m)$ such that $t(x)$ divides $p(x)$, where:*

$$p(x) = \left(v_0(x) + \sum_{k=1}^{m} c_k \cdot v_k(x)\right) \cdot \left(w_0(x) + \sum_{k=1}^{m} c_k \cdot w_k(x)\right) - \left(y_0(x) + \sum_{k=1}^{m} c_k \cdot y_k(x)\right) .$$

# ZKP - Verifiable Computation (VC)

Pinocchio - QAP, QSP, (ZK : randomize by adding delta * t(s))

**Definition 2 (Quadratic Arithmetic Program (QAP) [30])**
*A QAP $Q$ over field $\mathbb{F}$ contains three sets of $m+1$ polynomials $\mathcal{V} = \{v_k(x)\}$, $\mathcal{W} = \{w_k(x)\}$, $\mathcal{Y} = \{y_k(x)\}$, for $k \in \{0 \ldots m\}$, and a targ… takes as in… a total of … computes … F's inputs … $(c_{N+1}, \ldots,$…*

$$p(x) = \ldots - \left( y_0(x) + \sum_{k=1}^{m} c_k \cdot y_k(x) \right) .$$

*Choose* $s, \alpha, \beta_v, \beta_w, \beta_y, \gamma \xleftarrow{R} \mathbb{F}$.

*Construct the public evaluation key* $EK_F$ *as:*

$$\begin{array}{llll} ( & \{g^{v_k(s)}\}_{k \in I_{mid}}, & \{g^{w_k(s)}\}_{k \in [m]}, & \{g^{y_k(s)}\}_{k \in [m]}, \\ & \{g^{\alpha v_k(s)}\}_{k \in I_{mid}}, & \{g^{\alpha w_k(s)}\}_{k \in [m]}, & \{g^{\alpha y_k(s)}\}_{k \in [m]}, \\ & \{g^{\beta_v v_k(s)}\}_{k \in I_{mid}}, & \{g^{\beta_w w_k(s)}\}_{k \in [m]}, & \{g^{\beta_y y_k(s)}\}_{k \in [m]} \\ & \{g^{s^i}\}_{i \in [d]}, & \{g^{\alpha s^i}\}_{i \in [d]} & ). \end{array}$$

*The public verification key is:* $VK_F = (g^1, g^\alpha, g^\gamma, g^{\beta_v \gamma},$

# Block Chain

{
"hash": "0000000000000000000286d1cc5c70c4a….",
"confirmations": 1,
"strippedsize": 193260,
"size": 276539,
"weight": 856319,
"height": 600480,
"version": 549453824,
"versionHex": "20c00000",
"merkleroot": "3b8532757885d28….",
"tx": [
…
…
…
]
"time": 1571719493,
"mediantime": 1571717163,
"nonce": 3512210036,
}

**No forgery**

Hash

{
"hash": "0000000000000000000a12c764baa33c3154...",
"confirmations": 2,
"strippedsize": 367418,
"size": 430287,
"weight": 1532541,
"height": 600481,
…
}

# Block Chain

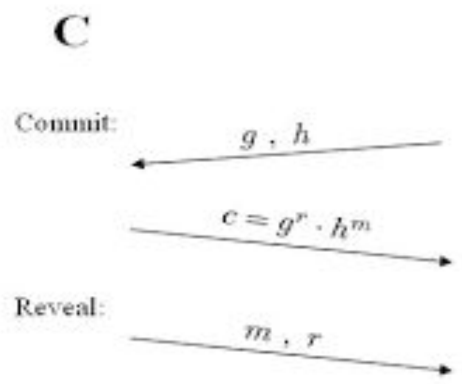
**Consensus**

- Proof of Work - bitcoin
  - 거래 확정에 많은 시간이 소요된다. (일시적인 fork)
  - 약 1시간의 시간이 거래 확정에 소요
- Proof of Sake - ethereum
  - 예치금의 방식으로 많은 돈을 예치하면 다음 블록 결정에 큰 영향을 줄 수 있음
- Byzantine agreement algorithm - algorand
  - No fork, but huge communication cost
  - To solve this problem, algorand uses VRF (verifiable random function)

# ZKP + Block Chain - Zcash, Hyperledger Indy

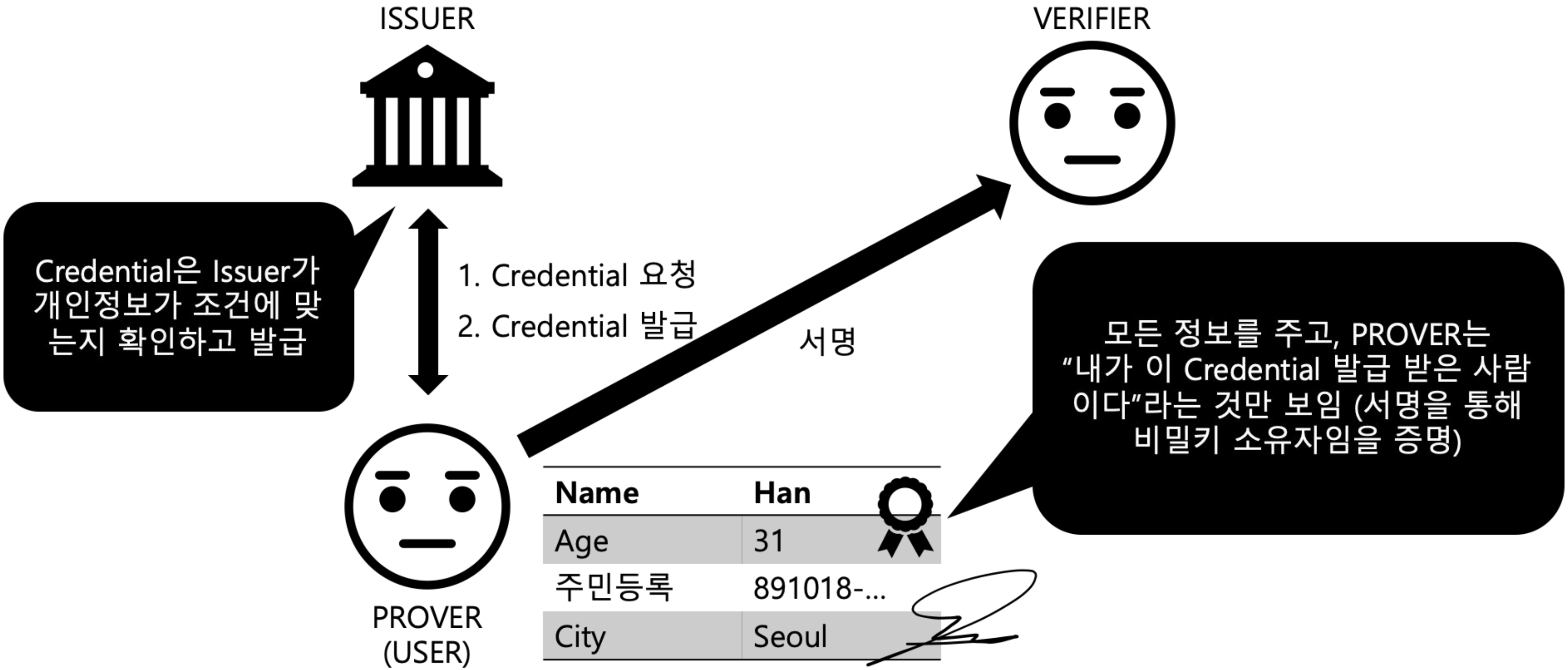
**ZKP != Privacy, ZKP == Honest Computation == 관계증명**

- Pedersen Commitment :

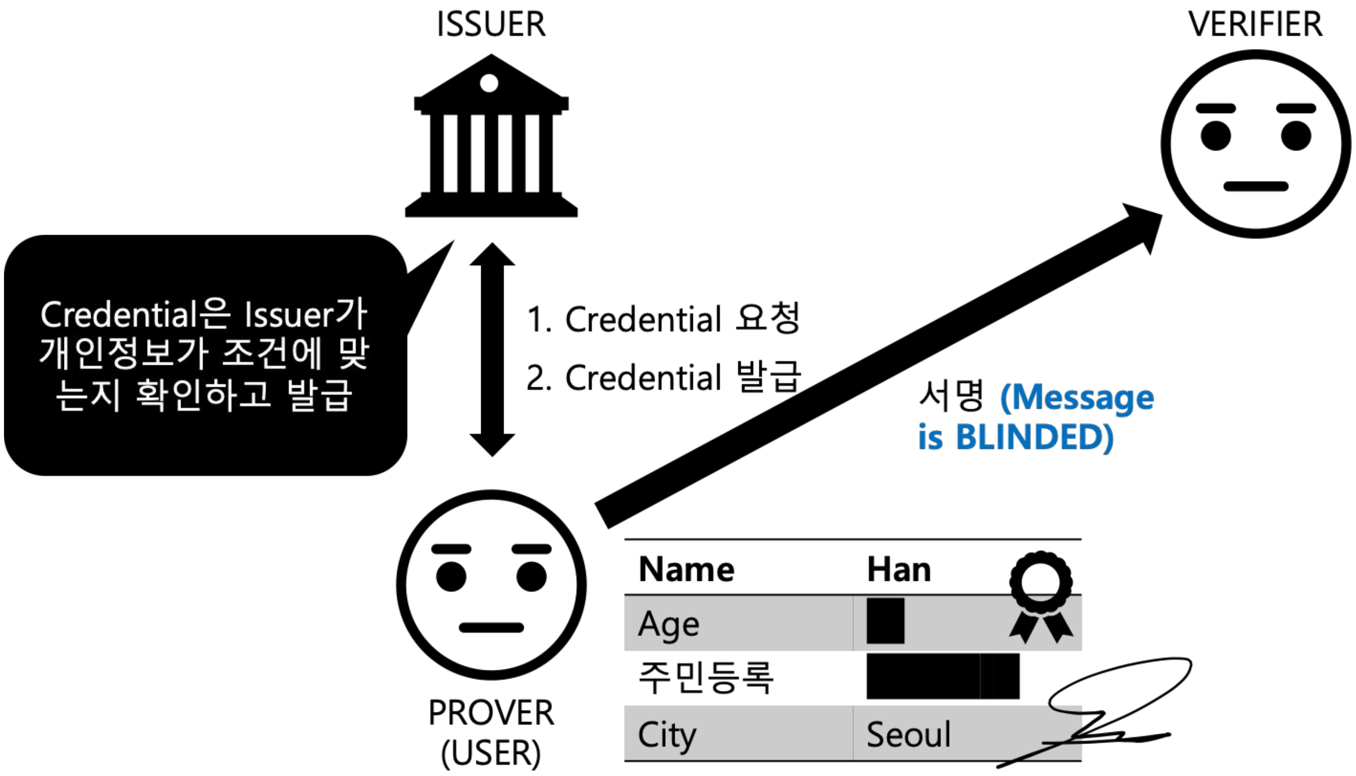



- Zcash: balance value is committed (=c) and prove that {c = comm(m) and m in some range}
- Indy: identity data is committed and sign on the committed value

# Hyperledger Indy



# Hyperledger



# Hyperledger - Verification

**Message** **Commitment** **Signature**

**Message** **Commitment** **Signature**

"Commitment가 Some Message에 의해서 생성되었다." 즉
나는 "c = Comm(m, r)"인 (m, r)을 알고있다.

**Message** **Commitment** **Signature**

"Signature가 Some Message에 의해서 생성되었다." 즉
나는 "s = Sign(m)"인 m을 알고있다.

# ZKP + Block Chain

## Example - proof of commitment

For given message m, let $c = g^m h^r$ for random r.

**Proof:**

1. Pick random x, y
2. $a = Hash(g^x h^y)$ - Fiat-Shamir
3. Compute $\pi 1 = x + am$ and $\pi 2 = y + ar$
4. Return ($\pi 1$, $\pi 2$, a)

**Verification:**

1. $c \overset{?}{=} Hash(g^{\pi 1} h^{\pi 2} / c^a)$

✝️ Proof of same secret, Proof of square, Proof of range, ... also possible