Implementing Searchable Encryption schemes over Multilinear Maps

Victor-Mihai Talif

Facultatea de Informatică, UAIC Iași

6 Iulie 2018

Definiție

- Datele în format original, necriptat păstrate virtual în medii cloud reprezintă o vulnerabilitate majoră pentru orice utilizator.

Definiție


INDY/TECH
DROPBOX HACK: CLOUD STORAGE COMPANY HACKED, POTENTIALLY REVEALING OVER 60 MILLION PASSWORDS

Definiție

- De aceea, datele sunt mereu stocate în format criptat.

Definiție

As cloud storage becomes more common, data security is an increasing concern. Companies and schools have been increasing their use of services like Google Drive for some time, and lots of individual users also store files on Dropbox, Box, Amazon Drive, Microsoft OneDrive and the like. They're no doubt concerned about keeping their information private – and millions more users might store data online if they were more certain of its security.

Data stored in the cloud is nearly always stored in an encrypted form that would need to be cracked before an intruder could read the information. But as a scholar of cloud computing and cloud security, I've seen that where the keys to that encryption are held varies among cloud storage services. In addition, there are relatively simple ways users can boost their own data's security beyond what's built into systems they use.

Definiție

- Dar cum putem efectua căutari pe multimi de date criptate?
- **Criptare deterministă față de caracteristicile conținutului? NU**. Orice informație relevată de criptare poate fi exploatată de atacatori.
  - **Criptarea trebuie să fie probabilistă.**
- **Decriptăm conținutul la nivelul serverului bazei de date? NU**. Manipularea datelor la nivelul bazei de date expune conținutul lor într-un mediu extern utilizatorului, vulnerabil la atacuri.
  - **Se dorește ca datele să fie decriptate doar în medii de încredere, precum entitatea master a criptosistemului sau chiar de către utilizator.**

Definiție

- Se caută menținerea unui echilibru între eficiența funcției de căutare și menținerea securității.

Definiție

- Astfel definim conceptul de **Searchable Encryption**:

Searchable Encryption reprezintă un criptosistem în care pot fi făcute interogări asupra datelor criptate (criptotextelor) în urma cărora să rezulte submulțimi ale bazei de date fara a fi necesară decriptarea.

Implementări

- Cea mai cunoscută implementare pentru această problemă este **Hidden Vector Encryption**.
- Criptotextelor le este asociat un vector de atribute predeterminate care să releve caracteristicile datelor criptate, fără a le dezvălui conținutul.
- Cu ajutorul acestor vectori se pot efectua interogări : **de egalitate**, dar și **comparative** și **de incluziune**.

Implementări

- Implementări abordate:
  - **Boneh-Waters** (2007)
  - **Iovino-Persiano** (2008)*
- Ambele folosesc aplicații biliniare.

NIKE

Definiție

- **Non-Interactive Key Exchange**. Permite mai multor utilizatori : stabilirea unei chei (unui secret) comune sau unei semnături digitale comune.
- Realizează acest lucru cu minimul de interacțiuni : odată cu stabilirea comunicării se face și schimbul de parametri necesari.
- Este sigur : dezvăluirea unei chei comune nu va afecta securitatea celorlalte chei (“no collusion"); dezvăluirea unei chei private nu va afecta securitatea cheilor împărtășite de utilizatori nedezvăluiți.

NIKE

Construcție

- Lucrăm în grupul G de ordin n, cu un generator g.
- Între 2 utilizatori, A_1 și A_2 : Fiecare utilizator își generează cheia publică $g^{a_i} \in G$ și cheia privată $a_i \in \mathbb{Z}_n$.
- Făcand schimb de chei publice, fiecare utilizator va genera cheia comună ridicand cheia publică primită la propria cheie privată:

$$(g^{a_1})^{a_2} = (g^{a_2})^{a_1} = g^{a_1 a_2}$$

NIKE

Construcție

- Lucrăm în grupul G de ordin n, cu un generator g.
- Între 3 utilizatori, A_1, A_2 și A_3 : Fiecare utilizator își generează cheia publică $g^{a_i} \in G$ și cheia privată $a_i \in \mathbb{Z}_n$.
- Făcand schimb de chei publice, fiecare utilizator trebuie să genereze cheia comună folosind cheile publice ale celorlalți 2 utilizatori și cheia sa privată.
- Cum face aceasta?

NIKE

Construcție

- **Aplicații biliniare**. O aplicație biliniară $e : G \times G \to G_T$ are următoarele proprietăți:
  - **Biliniaritate** : $\forall a, b \in \mathbb{Z}_n$, $e(g^a, g^b) = e(g, g)^{ab}$
  - **Nedegenerare** : $e(g, g) = g'$, unde g' este generator in G_T. Cum grupurile au același ordin, avem și $e(g^n, g) = e(g, g)^n = g'^n = 1$.
- Fiecare utilizator se va folosi deci de cheile publice ale celorlalți utilizatori ca parametri ai aplicației biliniare, pe care o va ridica la cheia sa privată pentru a obține cheia publică:

$$e(g^{a_1}, g^{a_2})^{a_3} = e(g^{a_3}, g^{a_1})^{a_2} = e(g^{a_2}, g^{a_3})^{a_1} = e(g, g)^{a_1 a_2 a_3}$$

- Implementări reușite : **Pairings**.

NIKE

Construcție

- Lucrăm în grupul G de ordin n, cu un generator g.
- Între N utilizatori, A_1, A_2,..,A_N : Fiecare utilizator își generează cheia publică $g^{a_i} \in G$ și cheia privată $a_i \in \mathbb{Z}_n$.
- Făcând schimb de chei publice, fiecare utilizator trebuie să genereze cheia comună folosind cheile publice ale celorlalti N-1 utilizatori și cheia sa privată.
- Cum face aceasta?

NIKE

Construcție

- **Aplicații multiliniare**. O aplicație multiliniară $e: G^k \to G_T$ are următoarele proprietăți:
  - **Multiniaritate** : $\forall a_1, a_2, .., a_k \in \mathbb{Z}_n$, $e(g^{a_1}, g^{a_2}, .., g^{a_k}) = e(g, g)^{a_1 a_2 \ldots a_k}$
  - **Nedegenerare** : $e(g, g, ..., g) = g'$, unde g' este generator in G_T. Cum grupurile au același ordin, avem și $e(g^n, g, ..., g) = e(g, g, ..., g)^n = g'^n = 1$.
- Fiecare utilizator se va folosi deci de cheile publice ale celorlalți utilizatori ca parametri ai aplicației multiliniare (N-1), pe care o va ridica la cheia sa privată pentru a obține cheia publică:

$$e(g^{a_1}, g^{a_2}, ..., g^{a_{N-1}})^{a_N} = e(g, g, ..., g)^{a_1 a_2 \ldots a_N}$$

- Implementări reușite : **Problemă deschisă**.

Contribuții

- Folosim schema Iovino-Persiano HVE. O extindem astfel încât să poată lucra cu :
  - Grupuri de **orice ordin**
  - **Aplicații multiliniare**
- O veritabilă generalizare a schemei.

Contribuții

- **Schema originală Iovino-Persiano.**

1. **Setup(λ, n).** The inputs received are security parameter λ and attribute length n. The algorithm generates a large prime number p. It then generates group G of order p. It then composes the instance $\mathcal{I} = (p, G, G_T, e)$.

   Afterwards, it randomly generates the following elements:

$$\boxed{y \in \mathbb{Z}_p}$$
$$\forall\, 1 \leq i \leq n \;:\; t_i, v_i, r_i, m_i \in \mathbb{Z}_p$$

   These all compose the masterkey **MsK**:

$$MsK = (y, (t_i, v_i, r_i, m_i)_{\forall i \in \overline{1,n}})$$

   It then generates the following values:

$$\boxed{Y = e(g, g)^y}$$
$$\forall\, 1 \leq i \leq n \;:$$
$$\boxed{\begin{aligned} T_i &= g^{t_i} \\ V_i &= g^{v_i} \\ R_i &= g^{r_i} \\ M_i &= g^{m_i} \end{aligned}}$$

   It then publishes the public key **PK** using the instance $\mathcal{I}$ and the previously determined elements:

$$PK = (\mathcal{I}, Y, (T_i, V_i, R_i, M_i)_{\forall i \in \overline{1,n}}))$$

Contribuții

- **Schema originală Iovino-Persiano**.

2. **Encrypt(PK,*(I,M)*)**. Given as input index $I = (I_1, I_2, ..., I_l) \in \Sigma^l$ and a message $M \in G_T$, the encryption algorithm works as such:

   (a) Select random $s \in \mathbb{Z}_p$ and random $s_1, s_2, ..., s_n \in \mathbb{Z}_p$.

   (b) Compute the following cypertext elements:

$$\Omega = M \cdot Y^{-s}$$

$$C_0 = g^s$$

$$\forall 1 \leq i \leq n:$$

$$X_i = \begin{cases} T_i^{s-s_i}, & \text{if } I_i = 1 \\ R_i^{s-s_i}, & \text{if } I_i = 0 \end{cases}$$

$$W_i = \begin{cases} V_i^{s_i}, & \text{if } I_i = 1 \\ M_i^{s_i}, & \text{if } I_i = 0 \end{cases}$$

Contribuții

- **Schema originală Iovino-Persiano.**

3. **GenToken(MsK, $\mathbf{I}_\star$).** Given predicate description $I_\star = (\mathcal{I}_1, \mathcal{I}_2, ..., \mathcal{I}_l) \in \Sigma_\star$, the algorithm determines the set S of indexes $1 \leq i \leq l$ such that $I_i \neq \star$.

   If $S = \emptyset$, that is, if $I_\star$ only has wildcard entries, the algorithm will output the key:

$$\boxed{K_0 = g^y}$$

   Otherwise, for each element i of S, we will randomly generate value a_i, such that in the end, we have:

$$\boxed{\sum_{i \in S} a_i = y}$$

   For each of these generated elements, we compute:

$$\boxed{\begin{aligned} Y_i &= \begin{cases} g^{\frac{a_i}{t_i}}, & \text{if } \mathcal{I}_i = 1 \\ g^{\frac{a_i}{r_i}}, & \text{if } \mathcal{I}_i = 0 \\ \emptyset, & \text{otherwise} \end{cases}, \\ L_i &= \begin{cases} g^{\frac{a_i}{v_i}}, & \text{if } \mathcal{I}_i = 1 \\ g^{\frac{a_i}{m_i}}, & \text{if } \mathcal{I}_i = 0 \\ \emptyset, & \text{otherwise} \end{cases} \end{aligned}}$$

Contribuții

- **Schema originală Iovino-Persiano**.

4. **Query(TK,C)**. Keeping the preceding notations and minding which of the situations we are in, we compute one of the following:

(a) $M \leftarrow \Omega \cdot e(C_0, K_0)$

(b) $M \leftarrow \Omega \cdot \prod_{i \in S} e(X_i, Y_i) e(W_i, L_i)$

M will be correct if the attribute vector satisfies the predicate in question.

Contribuții

- **Schema originală Iovino-Persiano.**

Proof. For our proof of query correctness, we keep the same notation used in the aforementioned construction, right down to the random index-message pair (I, M).

For case (a), we have :

$$\Omega \cdot e(C_0, K_0) = M \cdot e(g,g)^{-ys} \cdot e(g,g)^{ys} = M$$

For case (b), assuming the predicate is satisfied by the attribute vector, we have :

$$\Omega \cdot \prod_{i \in S} e(X_i, Y_i) e(W_i, L_i) = M \cdot e(g,g)^{-ys} \cdot \prod_{i \in S} e(g,g)^{(s-s_i)a_i} \cdot e(g,g)^{s_i a_i} =$$

$$M \cdot e(g,g)^{-ys} \cdot \prod_{i \in S} e(g,g)^{s a_i} = M \cdot e(g,g)^{-ys} \cdot e(g,g)^{s \cdot \sum_{i \in S} a_i} = M \cdot e(g,g)^{-ys} \cdot e(g,g)^{ys} = M$$

In our computation, regardless of whether $I_i = 1$ or $I_i = 0$, we reached:

$$(s - s_i) \cdot t_i \cdot \tfrac{a_i}{t_i} = (s - s_i) \cdot r_i \cdot \tfrac{a_i}{r_i} = (s - s_i) \cdot a_i$$

$$s_i \cdot v_i \cdot \tfrac{a_i}{v_i} = s_i \cdot m_i \cdot \tfrac{a_i}{m_i} = s_i \cdot a_i$$

Contribuții

- Elementele a_i formează o structură aditivă generată aleator (pentru suma fixată).

$$\prod_{i \in S} e(X_i, Y_i) e(W_i, L_i)$$

$$\Downarrow$$

$$e(X_1, X_2, ..., X_N, Y_1, Y_2, ..., Y_N) e(W_1, W_2, ..., W_N, L_1, L_2, ..., L_N)$$

- Păstrând ideea construcției originale, aici elementele a_i vor fi nevoite să formeze o structură multiplicativă generată aleator pentru valoarea y. Construcția originală fiind peste un grup de ordin prim, aceasta este realizabilă, deoarece orice exponent al unui element din grup își va găsi un invers multiplicativ.
- Sunt necesare și schimbări pentru exponenții s.

Contribuții

- **Schema multiliniară Iovino-Persiano, grupuri de ordin prim.**

1. **Setup(λ, n)**. The inputs received are security parameter λ and attribute length n. The algorithm generates a large prime number p. It then generates group G of order p. It then composes the instance $\mathcal{I} = (p, G, G_T, e)$, where $e : G^{2n} \to G_T$ is our multilinear map function.

   Afterwards, it randomly generates the following elements:

$$\boxed{y \in \mathbb{Z}_p}$$
$$\forall 1 \le i \le n \ : t_i, v_i, r_i, m_i \in \mathbb{Z}_p$$

   These all compose the masterkey **MsK**:

$$MsK = (y, (t_i, v_i, r_i, m_i)_{\forall i \in \overline{1,n}})$$

   It then generates the following values:

$$\boxed{Y = e(g, g, ..., g)^y}$$
$$\forall 1 \le i \le n \ :$$
$$\boxed{\begin{array}{c} T_i = g^{t_i} \\ V_i = g^{v_i} \\ R_i = g^{r_i} \\ M_i = g^{m_i} \end{array}}$$

   It then publishes the public key **PK** using the instance $\mathcal{I}$ and the previously determined elements:

$$PK = (\mathcal{I}, Y, (T_i, V_i, R_i, M_i)_{\forall i \in \overline{1,n}}))$$

Contribuții

- **Schema multiliniară Iovino-Persiano, grupuri de ordin prim.**

2. **Encrypt(PK,(*I*,*M*)).** Given as input index $I = (I_1, I_2, ..., I_l) \in \Sigma^l$ and a message $M \in G_T$, the encryption algorithm works as such:

   (a) Select random $s \in \mathbb{Z}_p$ and random $s' \in \mathbb{Z}_p$.

   (b) Compute the rows s_i and s'_i for $i \in \overline{1,n}$ such that :

   $$\prod_{i \in \overline{1,n}} s_i = s - s'$$
   $$\prod_{i \in \overline{1,n}} s'_i = s'$$

   The fact that this computation is possible will be addressed later, for the row generated for the *GenToken* function.

   (c) Compute the following cypertext elements:

   $$\Omega = M \cdot Y^{-s}$$
   $$C_0 = g^s$$
   $$\forall 1 < i < n :$$
   $$X_i = \begin{cases} T_i^{s_i}, & \text{if } I_i = 1 \\ R_i^{s_i}, & \text{if } I_i = 0 \end{cases}$$
   $$W_i = \begin{cases} V_i^{s'_i}, & \text{if } I_i = 1 \\ M_i^{s'_i}, & \text{if } I_i = 0 \end{cases}$$

Contribuții

- **Schema multiliniară Iovino-Persiano, grupuri de ordin prim.**

3. **GenToken(MsK, I$_\star$)**. Given predicate description $I_\star = (\mathcal{I}_1, \mathcal{I}_2, ..., \mathcal{I}_l) \in \Sigma_\star$, the algorithm determines the set S of indexes $1 \leq i \leq l$ such that $I_i \neq \star$.

If $S = \emptyset$, that is, if $I_\star$ only has wildcard entries, the algorithm will output the key:

$$\boxed{K_0 = g^y}$$

Otherwise, for each element i of S, we will randomly generate value a_i, such that in the end, we have:

$$\boxed{\prod_{i \in S} a_i = y}$$

Given the prime order of group G, it follows that having computed the product of all but the last element, we will be able to invert said product and multiply it by y, thus obtaining what is needed of us. Computationally, this is of similar complexity to the original implementation: instead of randomly generating $|S|-1$ values in G and determining the last one by deducing the current value from y, we randomly generate $|S|-1$ values in G and given our group's properties, we invert the result and multiply it by y to obtain our result. Best implementations of modular inversing achieve it in logarithmic time.

For each of these generated elements, we compute:

$$\boxed{Y_i = \begin{cases} g^{\frac{a_i}{t_i}}, & \text{if } \mathcal{I}_i = 1 \\ g^{\frac{a_i}{r_i}}, & \text{if } \mathcal{I}_i = 0 \\ g, & \text{otherwise} \end{cases}, \qquad L_i = \begin{cases} g^{\frac{a_i}{v_i}}, & \text{if } \mathcal{I}_i = 1 \\ g^{\frac{a_i}{m_i}}, & \text{if } \mathcal{I}_i = 0 \\ g, & \text{otherwise} \end{cases}}$$

We output TK as either K_0 or $(Y_i, L_i)_{i \in \overline{1,n}}$, depending on which of the aforementioned cases is at hand.

Contribuții

- **Schema multiliniară Iovino-Persiano, grupuri de ordin prim**.

4. **Query(TK,C)**. Keeping the preceding notations and minding which of the situations we are in, we compute one of the following:

(a) $M \leftarrow \Omega \cdot e(C_0, K_0, g, g, ..., g)$

(b) $M \leftarrow \Omega \cdot e(X_1, X_2, ..., X_n, Y_1, Y_2, ..., Y_n) \cdot e(W_1, W_2, ..., W_n, L_1, L_2, ..., L_n)$

M will be correct if the attribute vector satisfies the predicate in question.

Contribuții

- **Schema multiliniară Iovino-Persiano, grupuri de ordin prim.**

Proof. For case (a), we have :

$$\Omega \cdot e(C_0, K_0, g, g, ..., g) = M \cdot e(g, g, ..., g)^{-ys} \cdot e(g^s, g^y, ..., g) =$$
$$M \cdot e(g, g, ..., g)^{-ys} \cdot e(g, g, ..., g)^{ys} = M$$

For case (b), assuming the predicate is satisfied by the attribute vector and denoting the rows:

$$\alpha_i = \begin{cases} t_i, & \text{if } I_i = 1 \\ r_i, & \text{if } I_i = 0 \end{cases}$$

$$\beta_i = \begin{cases} v_i, & \text{if } \mathcal{I}_i = 1 \\ m_i, & \text{if } \mathcal{I}_i = 0 \end{cases}$$

We use this notation for ease of writing, as if the predicate is satisfied, the values, regardless of the attribute vector entry, will be reduced. Therefore:

$$\Omega \cdot e(X_1, X_2, ..., X_n, Y_1, Y_2, ..., Y_n) \cdot e(W_1, W_2, ..., W_n, L_1, L_2, ..., L_n) =$$
$$=$$
$$\Omega \cdot e(g^{\alpha_1 s_1}, g^{\alpha_2 s_2}, ..., g^{\alpha_n s_n}, g^{\frac{a_1}{\alpha_1}}, g^{\frac{a_2}{\alpha_2}}, ..., g^{\frac{a_n}{\alpha_n}}) \cdot e(g^{\beta_1 s'_1}, g^{\beta_2 s'_2}, ..., g^{\beta_n s'_n}, g^{\frac{a_1}{\beta_1}}, g^{\frac{a_2}{\beta_2}}, ..., g^{\frac{a_n}{\beta_n}}) =$$
$$= \Omega \cdot e(g, g, ..., g)^{\prod_{i \in \overline{1,n}} s_i a_i} \cdot e(g, g, ..., g)^{\prod_{i \in \overline{1,n}} s'_i a_i} =$$
$$= \Omega \cdot e(g, g, ..., g)^{(s-s')y} \cdot e(g, g, ..., g)^{s'y} = \Omega \cdot e(g, g, ..., g)^{sy} =$$
$$= M \cdot e(g, g, ..., g)^{-ys} \cdot e(g, g, ..., g)^{ys} = M$$

Contribuții

- Extensia schemei pentru grupuri de ordin compus N, păstrând structura construcțiilor precedente, prezintă următoarele impasuri:
  - **Elemente inversabile**. În grupurile multiplicative de ordin compus, nu toate elementele sunt inversabile. Doar elementele coprime cu ordinul.
  - **Generarea structurii multiplicative**. Din același motiv, anume cel al elementelor inversabile, structura multiplicativă are nevoie de restricții.
- Soluția? **Restrângerea mulțimii $\mathbb{Z}_N$ la $\mathbb{Z}_N^\star$ pentru asigurarea elementelor inversabile.**
- Luand în calcul factorizarea lui N (factori primi foarte mari), reducerea de cardinalitate a mulțimii de exponenți este ignorabilă (99% rămâne).

Contribuții

- **Schema biliniară Iovino-Persiano, grupuri de ordin compus.**

1. **Setup(λ, n)**. The inputs received are security parameter λ and attribute length n. The algorithm generates $N = pq$, where p and q are large primes. It then generates group G of order N. It then composes the instance $\mathcal{I} = (p, G, G_T, e)$.

   Afterwards, it randomly generates the following elements:

$$y \in \mathbb{Z}_N$$
$$\boxed{\forall\, 1 \le i \le n \ : t_i, v_i, r_i, m_i \in \mathbb{Z}_N^*}$$

   These all compose the masterkey **MsK**:

$$MsK = (y, (t_i, v_i, r_i, m_i)_{\forall i \in \overline{1,n}})$$

   It then generates the following values:

$$\begin{gathered} Y = e(g,g)^y \\ \forall\, 1 \le i \le n \ : \\ T_i = g^{t_i} \\ V_i = g^{v_i} \\ R_i = g^{r_i} \\ M_i = g^{m_i} \end{gathered}$$

   It then publishes the public key **PK** using the instance $\mathcal{I}$ and the previously determined elements:

$$PK = (\mathcal{I}, Y, (T_i, V_i, R_i, M_i)_{\forall i \in \overline{1,n}}))$$

Contribuții

- **Schema multiliniară Iovino-Persiano, grupuri de ordin compus**.

1. **Setup(λ, n)**. The inputs received are security parameter λ and attribute length n. The algorithm generates $N = \prod_{i \in \overline{1,n}} p_i$, where all p_i are large prime factors. It then generates group G of order N. It then composes the instance $\mathcal{I} = (p, G, G_T, e)$, where $e : G^{2n} \to G_T$ is our multilinear map function.

   Afterwards, it randomly generates the following elements:

$$\boxed{y \in \mathbb{Z}_N^\star}$$
$$\boxed{\forall 1 \le i \le n \ : t_i, v_i, r_i, m_i \in \mathbb{Z}_N^\star}$$

   These all compose the masterkey **MsK**:

$$MsK = (y, (t_i, v_i, r_i, m_i)_{\forall i \in \overline{1,n}})$$

   It then generates the following values:

$$\begin{gathered} Y = e(g, g, ..., g)^y \\ \forall 1 \le i \le n \ : \\ T_i = g^{t_i} \\ V_i = g^{v_i} \\ R_i = g^{r_i} \\ M_i = g^{m_i} \end{gathered}$$

Contribuții

- **Schema multiliniară Iovino-Persiano, grupuri de ordin compus**.

2. **Encrypt(PK,(*I*,*M*))**. Given as input index $I = (I_1, I_2, ..., I_l) \in \Sigma^l$ and a message $M \in G_T$, the encryption algorithm works as such:

(a) Select random $s \in \mathbb{Z}_N^\star$ and random $s' \in \mathbb{Z}_N^\star$.

(b) Compute the rows s_i and s'_i in $\mathbb{Z}_N^\star$ for $i \in \overline{1,n}$ such that :

$$\prod_{i \in \overline{1,n}} s_i = s - s'$$
$$\prod_{i \in \overline{1,n}} s'_i = s'$$

(c) Compute the following cypertext elements:

$$\Omega = M \cdot Y^{-s}$$
$$C_0 = g^s$$
$$\forall\, 1 \leq i \leq n \ :$$
$$X_i = \begin{cases} T_i^{s_i}, & \text{if } I_i = 1 \\ R_i^{s_i}, & \text{if } I_i = 0 \end{cases}$$
$$W_i = \begin{cases} V_i^{s'_i}, & \text{if } I_i = 1 \\ M_i^{s'_i}, & \text{if } I_i = 0 \end{cases}$$

Contribuții

- **Schema multiliniară Iovino-Persiano, grupuri de ordin compus.**

3. **GenToken(MsK, I$_\star$).** Given predicate description $I_\star = (\mathcal{I}_1, \mathcal{I}_2, ..., \mathcal{I}_l) \in \Sigma_\star$, the algorithm determines the set S of indexes $1 \leq i \leq l$ such that $I_i \neq \star$.

   If $S = \emptyset$, that is, if $I_\star$ only has wildcard entries, the algorithm will output the key:

$$K_0 = g^y$$

Otherwise, for each element i of S, we will randomly generate value a_i in $\mathbb{Z}_N^\star$, such that in the end, we have:

$$\prod_{i \in S} a_i = y$$

For each of these generated elements, we compute:

$$Y_i = \begin{cases} g^{\frac{a_i}{t_i}}, & \text{if } \mathcal{I}_i = 1 \\ g^{\frac{a_i}{r_i}}, & \text{if } \mathcal{I}_i = 0 \\ g, & \text{otherwise} \end{cases},$$

$$L_i = \begin{cases} g^{\frac{a_i}{v_i}}, & \text{if } \mathcal{I}_i = 1 \\ g^{\frac{a_i}{m_i}}, & \text{if } \mathcal{I}_i = 0 \\ g, & \text{otherwise} \end{cases}$$

We output TK as either K_0 or $(Y_i, L_i)_{i \in \overline{1,n}}$, depending on which of the aforementioned cases is at hand.

Ce urmează?

- Continuarea cercetării ar presupune:
  - Demonstrații de securitate mai specifice sau avansate.
  - Extinderea schemei Boneh-Waters.
  - Identificarea unei soluții pentru construcția aplicațiilor multiliniare eficiente.