Practical Secure Aggregation for Privacy-Preserving Machine Learning (Ch 3.2, 3.4)

Insung Kong

Seoul National University

Department of Statistics

July 29, 2020

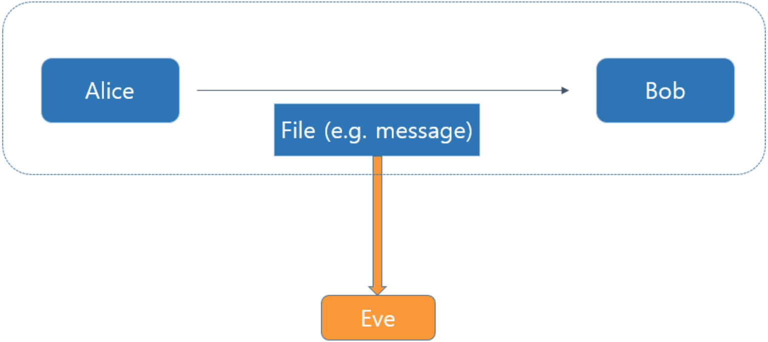
Overview

Section 1

3.2 Key Agreement

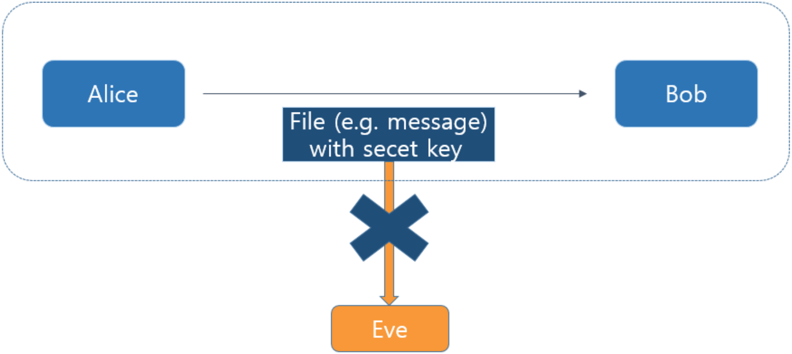
Key Agreement

- **A**lice want to send some file to **B**ob, but **adversaries E**ve can intercept file.



Key Agreement

- If they have **beforehand shared key**, they can protect content of file.
- But It's impossible to prepare shared key for all paired people.
- **Key Agreement** can make shared secret key safely through unsafe channel.



Key Agreement example - Diffie-Hellman key exchange

1. Alice and Bob determine prime **p**, and $\mathbf{g} \in \{1, 2, ..., p-1\}$ together.
   Eve can intercept it.
2. Alice makes secret private key **a**, Bob makes secret private key **b**.
   Eve can't intercept it.
3. Alice makes her public key $\mathbf{A} \equiv g^a (\text{mod p})$ and shares it.
   Bob makes his public key $\mathbf{B} \equiv g^b (\text{mod p})$ and shares it.
   Eve can intercept it.
4. Alice calculate $B^a \pmod{\text{p}}$, Bob calculate $A^b \pmod{\text{p}}$.
   Then they get **shared secret key** $g^{ab} \pmod{\text{p}}$.

- If p, a, b are large enough, Eve can't find shared secret key

Terms in this section

- **Adversary** : malicious entity whose aim is to prevent the users of the cryptosystem from achieving their goal
  - **Passive adversary** : one that can listen to your communications, but cannot directly tamper with them.
  - **Active adversary** : one that can listen to your communications, and also can directly tamper with them.
  - **PPT adversary** : an adversary who runs in probabilistic polynomial time algorithm.

- **Hash function** : it converts data with any length to fixed length data.
  - $\mathbf{H} : \{\mathbf{0}, \mathbf{1}\}^{\circ} \rightarrow \{\mathbf{0}, \mathbf{1}\}^{\mathbf{k}}$.

Key Agreement - Structure

Key Agreement consists of a tuple of algorithms :
(KA.param, KA.gen, KA.agree).

- **KA.param**$(k) \to pp$ produces some public parameters,
- **KA.gen**$(pp) \to (s_u^{SK}, s_u^{PK})$ allows any user u to generate a private-public key pair,
- **KA.agree**$(s_u^{SK}, s_v^{PK}) \to s_{u,v}$ allows any user u to combine their private key s_u^{SK} with the public key s_v^{PK} for any v.

Key Agreement - Structure

In the Diffie-Hellman key exchange,

- **KA.param**$(k) \rightarrow (\mathbb{G}', g, q, H)$ samples group $\mathbb{G}'$ of prime order q, a generator g, and a hash function $H : \{0,1\}^{\circ} \rightarrow \{0,1\}^{k}$
- **KA.gen**$(\mathbb{G}', g, q, H) \rightarrow (x, g^{x})$ samples a random $x \in \mathbb{G}'$ as the secret key s_u^{SK}, and g^x as the public key s_u^{PK}
- **KA.agree**$(x_u, g^{x_v}) \rightarrow s_{u,v}$ outputs $s_{u,v} = H((g^{x_v})^{x_u})$.

Definition 3.1 (Decisional Diffie-Hellman assumption)

We want that following two probability distributions are computationally indistinguishable in polynomial time.

- (g^a, g^b, g^{ab}), where a and b are randomly and independently chosen from $\mathbb{Z}_q$
- (g^a, g^b, g^c), where a, b, c are randomly and independently chosen from $\mathbb{Z}_q$

Definition 3.1 (Decisional Diffie-Hellman assumption)

Let $\mathcal{G}(\mathbf{k}) \rightarrow (\mathbb{G}', \mathbf{g}, \mathbf{q}, \mathbf{H})$ be an efficient algorithm which samples a group $\mathbb{G}'$ of order $\mathbf{q}$ with generator $\mathbf{g}$, as well as a function $\mathbf{H} : \{\mathbf{0}, \mathbf{1}\}^{\circ} \rightarrow \{\mathbf{0}, \mathbf{1}\}^{\mathbf{k}}$.
Consider the following probabilistic experiment, parameterized by a PPT adversary $\mathbf{M}$, and a security parameter $\mathbf{k}$.

$\mathbf{DDH} - \mathbf{Exp}_{\mathcal{G},\mathbf{M}}(\mathbf{k})$:

1. $(\mathbb{G}', g, q, H) \leftarrow \mathcal{G}(k)$
2. $a \leftarrow \mathbb{Z}_q$; $A \leftarrow g^a$
3. $b \leftarrow \mathbb{Z}_q$; $B \leftarrow g^b$
4. $e \stackrel{\$}{\leftarrow} \{0, 1\}$. if e=1, $s \leftarrow H(g^{ab})$, else $s \stackrel{\$}{\leftarrow} \{0, 1\}^k$
5. $M(\mathbb{G}', g, q, H, A, B, s) \rightarrow e'$
6. Output 1 if e=e' , 0 otherwise.

Definition 3.1 (Decisional Diffie-Hellman assumption)

$\mathbf{DDH - Exp_{\mathcal{G},M}(k)}$:

1. $(\mathbb{G}', g, q, H) \leftarrow \mathcal{G}(k)$
2. $a \leftarrow \mathbb{Z}_q$; $A \leftarrow g^a$
3. $b \leftarrow \mathbb{Z}_q$; $B \leftarrow g^b$
4. $e \stackrel{\$}{\leftarrow} \{0,1\}$. if e=1, $s \leftarrow H(g^{ab})$, else $s \stackrel{\$}{\leftarrow} \{0,1\}^k$
5. $M(\mathbb{G}', g, q, H, A, B, s) \rightarrow e'$
6. Output 1 if $e = e'$, 0 otherwise.

The advantage of the adversary is defined as

$$Adv^{DDH}_{\mathcal{G},M}(k) := 2\Pr\left[DDH - Exp_{\mathcal{G},M}(k) = 1\right] - 1$$

We say that the Decisional Diffie-Hellman assumption holds for $\mathcal{G}$ if for all PPT adversaries M, there exists a negligible function ϵ such that $Adv^{DDH}_{\mathcal{G},M}(k) \leq \epsilon(k)$

Definition 3.2 (Two Oracle Diffie-Hellman assumption)

- To prove security against **active adversaries**, we need a somewhat **stronger security guarantee** for Key Agreement.
- Assume adversary can get **public keys** s_u^{PK} and s_v^{PK} , and also have the ability to learn KA.agree(s_u^{SK} , s) and KA.agree(s_v^{SK} , s) for any $s \neq s_u^{PK}, s_v^{PK}$.
- We want this adversary still **cannot distinguish** $s_{u,v}$ from a random string.

Definition 3.2 (Two Oracle Diffie-Hellman assumption)

Let $\mathcal{G}(\mathbf{k}) \to (\mathbb{G}', \mathbf{g}, \mathbf{q}, \mathbf{H})$ be an efficient algorithm which samples a group $\mathbb{G}'$ of order $\mathbf{q}$ with generator $\mathbf{g}$, as well as a function $\mathbf{H} : \{\mathbf{0}, \mathbf{1}\}^{\circ} \to \{\mathbf{0}, \mathbf{1}\}^{\mathbf{k}}$.
Consider the following probabilistic experiment, parameterized by a PPT adversary $\mathbf{M}$, and a security parameter $\mathbf{k}$.

$\mathbf{2ODH} - \mathbf{Exp}_{\mathcal{G},\mathbf{M}}(\mathbf{k})$:

1. $(\mathbb{G}', g, q, H) \leftarrow \mathcal{G}(k)$; $a \leftarrow \mathbb{Z}_q$; $A \leftarrow g^a$; $b \leftarrow \mathbb{Z}_q$; $B \leftarrow g^b$
2. $e \xleftarrow{\$} \{0, 1\}$. if e=1, $s \leftarrow H(g^{ab})$, else $s \xleftarrow{\$} \{0, 1\}^k$
3. $M^{\mathcal{O}_a(\cdot), \mathcal{O}_b(\cdot)}(\mathbb{G}', g, q, H, A, B, s) \to e'$
4. Output 1 if $e = e'$, 0 otherwise.

where $\mathcal{O}_a(X)$ returns $H(X^a)$ on any $X \neq B$(and an error on input B), and $\mathcal{O}_b(X)$ returns $H(X^b)$ on any $X \neq A$(and an error on input A)

Definition 3.2 (Two Oracle Diffie-Hellman assumption)

$\mathbf{2ODH - Exp_{\mathcal{G},M}(k)}$:

1. $(\mathbb{G}', g, q, H) \leftarrow \mathcal{G}(k)$; $a \leftarrow \mathbb{Z}_q$; $A \leftarrow g^a$; $b \leftarrow \mathbb{Z}_q$; $B \leftarrow g^b$
2. $e \xleftarrow{\$} \{0,1\}$. if e=1, $s \leftarrow H(g^{ab})$, else $s \xleftarrow{\$} \{0,1\}^k$
3. $M^{\mathcal{O}_a(\cdot),\mathcal{O}_b(\cdot)}(\mathbb{G}', g, q, H, A, B, s) \rightarrow e'$
4. Output 1 if $e = e'$, 0 otherwise.

where $\mathcal{O}_a(X)$ returns $H(X^a)$ on any $X \neq B$(and an error on input B), and $\mathcal{O}_b(X)$ returns $H(X^b)$ on any $X \neq A$(and an error on input A)

The advantage of the adversary is defined as

$$Adv_{\mathcal{G},M}^{2ODH}(k) := 2\Pr\left[2ODH - Exp_{\mathcal{G},M}^{1}(k) = 1\right] - 1$$

We say that the Two Oracle Diffie-Hellman assumption holds for $\mathcal{G}$ if for all PPT adversaries M, there exists a negligible function ϵ such that $Adv_{\mathcal{G},M}^{DDH}(k) \leq \epsilon(k)$

Section 2

3.4 Pseudorandom Generator

Pseudorandom Generator

- **PRG**(Pseudorandom Generator) takes in a uniformly random seed of some fixed length, and whose output space is $[0, R)^m$.
- Security for a **PRG** guarantees that its output is **computationally indistinguishable** from a **uniformly sampled** element of the output space.

Pseudorandom Generator

Linear Congruential Generator is most common and oldest algorithm for generating pseudo-randomized numbers.
The generator is defined by the recurrence relation:

Linear Congruential Generator

$X_{n+1} = aX_n + c$ (mod R)

where X is the sequence of pseudo-random values,

$m, \quad 0 < R$ - modulus

$a, \quad 0 < a < R$ - multiplier

$c, \quad 0 < c < R$ - increment

$x_0, \quad 0 \leq x_0 < R$ - the seed or start value