Revision Notes for COM3030: Privacy Enhancing Technologies 2024/25

Pascal U

University of Surrey

Abstract. Revision notes covering weeks 7 through 10. Covering all topics as presented in lectures, going into further detail as needed and ignoring some sections if not essential. Some (potentially many) parts could be wrong in detail, but certainly not for a lack of trying to understand the concepts deeply.

1 Week 7: Secure Messaging

1.1 Security of Encryption Schemes

IND-CPA Security The adversary chooses two plaintext, one of them is encrypted, and they cannot query a decryption oracle.

- The adversary $\mathcal{A}$ chooses two plaintexts. One of them is encrypted, and the adversary cannot query a decryption oracle.
- The challenger runs $(\text{pk}, \text{sk}) \leftarrow \text{KeyGen}(\lambda)$.
- $\mathcal{A}$ submits two plaintexts m_0 and m_1.
- The challenger picks a random bit $b \in \{0, 1\}$ and encrypts m_b to obtain $c \leftarrow \text{Enc}(\text{pk}, m_b)$.
- $\mathcal{A}$ receives the ciphertext c.
- $\mathcal{A}$ guesses b'.
- If $\mathcal{A}$ is unable to distinguish b with a non-negligible advantage over random guessing (50% chance), the encryption scheme is considered **IND-CPA secure**.

$$\Pr[b' = b] - \frac{1}{2} \approx 0,$$

IND-CCA Security A stronger security requirement whereby adversary is able to query a decryption oracle.

- The challenger runs $(\text{pk}, \text{sk}) \leftarrow \text{KeyGen}(\lambda)$.
- The adversary $\mathcal{A}$ submits two plaintexts m_0 and m_1.
- The challenger picks a random bit $b \in \{0, 1\}$ and encrypts m_b to obtain $c^* \leftarrow \text{Enc}(\text{pk}, m_b)$.
- $\mathcal{A}$ receives the ciphertext c^*.
- $\mathcal{A}$ is granted access to a decryption oracle $\text{Dec}(\text{sk}, *)$, which decrypts any ciphertext $c \neq c^*$.

- $\mathcal{A}$ guesses b'.
- If $\mathcal{A}$ is unable to distinguish b with a non-negligible advantage over random guessing (50% chance), the encryption scheme is considered **IND-CCA secure**.

$$\Pr[b' = b] - \frac{1}{2} \approx 0,$$

1.2 Digital Signatures

- **KeyGen**: Takes input λ (security parameter) and outputs a key pair (sk, vk).
- **Sign**: Takes input message m, a signing key sk, and outputs a signature σ.
- **Verify**: Takes signature σ, a verification key vk, a message m, and outputs 1 (if valid) or 0 (if invalid).

Forgery Security

- **Universal Unforgeability (UUF)**: The adversary is able to create a forged signature on a given random message.
- **Selective Unforgeability (SUF)**: The adversary is able to create a forged signature on a message they choose before the attack.
- **Existential Unforgeability (EUF)**: The adversary is able to create a forged signature on any message they choose, provided it hasn't been signed by the legitimate signer.

Attack Security

- **Key Only Attack (KOA)**: The adversary knows the verification key, but nothing else.
- **Known Message Attack (KMA)**: The adversary knows the verification key and some signatures on messages for which it knows the content but hasn't chosen.
- **Chosen Message Attack (CMA)**: The adversary knows the verification key and can obtain the signature on any set of messages it chooses.

EUF-CMA

- The adversary $\mathcal{A}$ can make queries to a signing oracle. For each message m, $\mathcal{A}$ can request a signature $\sigma \leftarrow \text{Sign}(\text{sk}, m)$. (Satisfies CMA)
- After querying for signatures on any number of messages, $\mathcal{A}$ must output a new message m' and a signature σ'.
- A forgery is considered valid if:

$$\text{Verify}(\text{pk}, m', \sigma') = 1$$

(Satisfies EUF if $m' \notin$ all queried m)

1.3 Encryption Methods and Protocols for Data Transit

Authenticated Encryption with Associated Data (AEAD) Scheme

- **KeyGen**: Takes input λ (security parameter) and outputs a key pair (sk, pk).
- **Encrypt**: Takes input a message m, a header hdr, and a public key pk, and outputs a ciphertext c and a tag μ.
- **Decrypt**: Takes input a ciphertext c, a header hdr, and a secret key sk, and outputs a message m if and only if sk is correct and μ verifies. Otherwise, it outputs an error.

An AEAD scheme is secure if it offers both IND-CCA and EUF-CMA security.

Key Derivation Function (KDF)

- **KDF(seed, l, salt, c)**: A deterministic algorithm that takes as input a seed, output length l, a salt, and a context variable c. The salt and context variable are optional and are usually ignored. The output is a cryptographic key k.
- **Security Properties**: A KDF is secure if an attacker, with knowledge of a key k (and the context and output length), cannot recover the seed (or the salt).

KDF Chains The output of one KDF function is the input to the next KDF function in the chain.

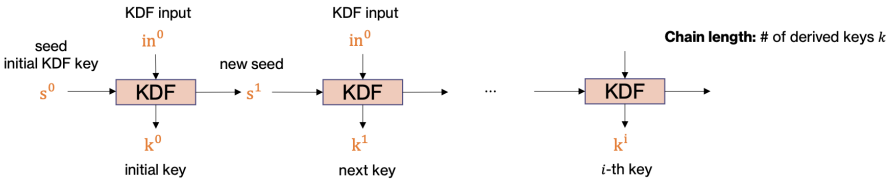


Fig. 1. KDF Chain

- Seeds are used only as KDF keys and are deleted after use.
- KDF inputs can be constant or change, as this does not impact key secrecy.
- All AEAD operations are performed using only derived keys k^i.
- If the adversary learns s^i or k^i, they cannot compute previous seeds/keys (forward secrecy). However, this does not protect against post-compromise security.

Symmetric Key Ratchet with KDF Chains Given two participants, Alice and Bob maintain different KDF chains for:

1. **Sending**: Encrypting outgoing messages, where Alice's sending chain = Bob's receiving chain.
2. **Receiving**: Decrypting incoming messages, where Alice's receiving chain = Bob's sending chain.

The Double Ratchet System Provides forward secrecy by ongoing renewal and maintenance of short-lived session keys. Through combination of two Symmetric Key Ratchets and a Public Key Ratchet for each participant.

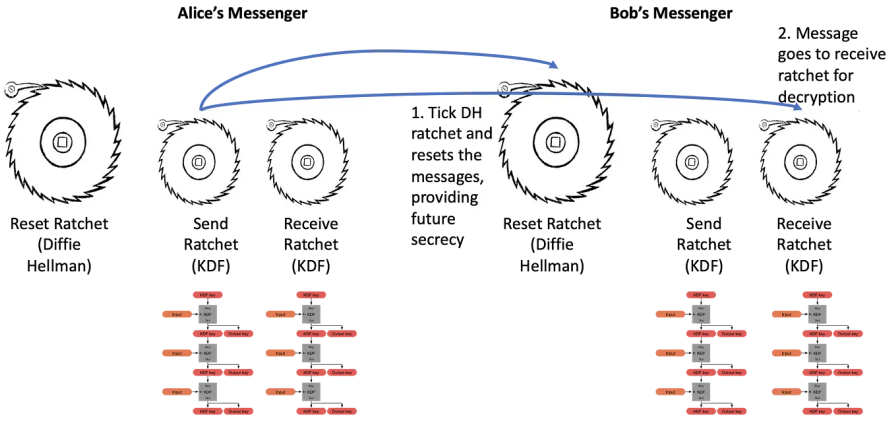


Fig. 2. KDF Chain

– **KDF**: Used for Symmetric Key Ratcheting.
– **Diffie-Hellman**: Used for Public Key Ratcheting.

Out-of-order messages are handled by a separate algorithm:

– **Variables:**
  • N: Message counter in the current sending chain.
  • P: Length of the previous sending chain.
– **Algorithm:**
  1. If a received message does not cause a PK ratchet:
     • The number of messages missed in the current receive chain is:

$$\text{Missed Messages} = N - \text{length(receive)}.$$

  2. Else, if a PK ratchet occurs:

- The number of messages missed in the current receive chain is:

$$\text{Missed Messages} = P - \text{length(receive)}.$$

3. Derive all the missing keys.
4. Decrypt the message m, but store the skipped keys for later use.

Trust in server infrastructure is required because;

- Server can drop messages (i.e. DoS).
- Server can deliberately delay messages to make users store secret keys longer than required.
- Server may accumulate messages with the goal to compromise them later.
- Replace (sk, pk) of some user to impersonate them by creating pre-key bundles with $\text{pre}_i = (i, \text{pk}_i, \sigma_i)$ (mitigated by physical or out-of-band verification).

Authenticated Key Exchange (AKE) Diffie Hellman (DH)

- take a generator g of a finite group with order p
- **Alice** samples her key pairs as $(sk_A, pk_A) = (a, A = g^a)$
- **Bob** samples his key pairs as $(sk_B, pk_B) = (b, B = g^b)$
- Consider a shared key g^{ab}
  - Alice can compute this as

$$(pk_B)^{sk_A} = (B)^a$$

  - Bob can compute this as

$$(pk_A)^{sk_B} = (A)^b$$

X3DH AKE protocol The handshake protocol used in Signal's E2EE messaging that relies on DH keys and provides deniability.

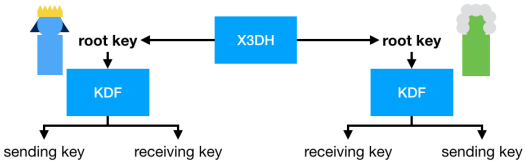


Fig. 3. X3DH handshake

- **Initial Setup:**
  - Alice and Bob each generate long-term and ephemeral key pairs (public and private)

- **Exchange of Public Keys:**
  - Alice sends Bob: epk_A, pk_A
  - Bob sends Alice: epk_B, pk_B
- **Key Agreement(DH Exchanges)** The shared secret is computed using:
  - **DH1:** Alice Computes DH(esk_A, pk_B)
  - **DH2:** Bob Computes DH(esk_B, pk_A)
  - **DH3:** Alice Computes DH(sk_A, epk_B)
- **Shared Secret Derivation:**

$$\text{shared\_secret} = \text{HKDF}(\text{DH1} \| \text{DH2} \| \text{DH3})$$

- **Session Key Agreement:**
  - Using shared_secret, Alice and Bob derive the session keys they will use to encrypt and decrypt messages in their communication.
  - The way this is done is not too important as long as they are consistent and secure on both ends.

1.4 Privacy and Security properties of the Signal Protocol

Post-Compromise Security of the Signal Protocol Key rotation, in combination with the symmetric ratchet, ensures that even if a key is compromised, future messages are protected by new keys.

- Ephemeral Keys:
  - Ratcheted regularly, therefore compromised keys cannot derivate future keys without access to Long-Term Secret Key.
- Session Keys:
  - Remember from **KDF Chains** that if the i^{th} key is compromised, past keys cannot be derived.
  - Therefore prior communication is still safe.
- Long-Term Keys:
  - If compromised, attacker may impersonate user, but cannot immediately decrypt messages due to ephemeral keys.

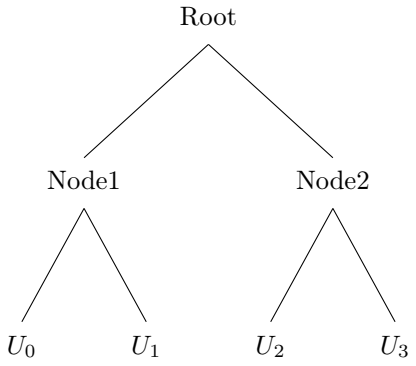
Deniability with AKE Any two users could have generated the protocol because the shared DH key can be individually generated by either party. This is known as **Offline Deniability**. **Online deniability** is not yet achieved in the described protocol. Because ephemeral keys must match for the message to be authenticated.

2 Week 8: Private Function Evaluation

2.1 Group Messaging and Open MLS

Messaging Layer Security (MLS) Is an approach at standardizing a solution to secure group messaging, at the core of MLS is the TreeKEM protocol. TreeKEM continuously generates fresh, shared, and secret randomness used by participants to evolve the group key. Each new group key is used to initiate a fresh symmetric hash ratchet that defines a streeam of nonce/key pairs used to symmetrically encrypt/decrypt messages using an AEAD. a stream is used until the next evolution of the group key at which point a new stream is initiated.

Group Creation with TreeKEM

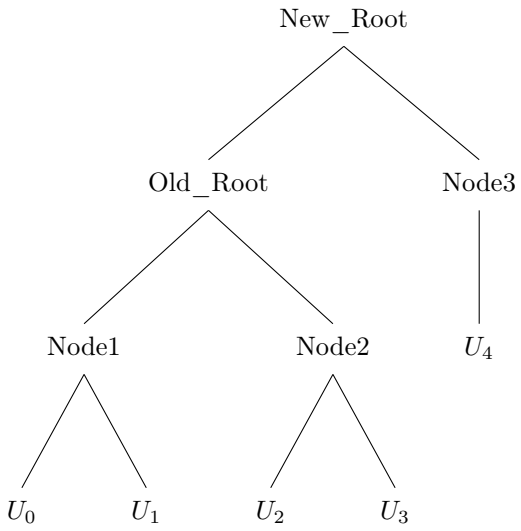


- Given a group of users, e.g.; G = (U_0, U_1, U_2, U_3)
- a ratchet tree (RT) is first created with the use of a Public Key Encryption Scheme (PKE), each user creates a key pair (pk_i, sk_i) corresponding to their respective identity U_i.
- Creator U_0 arranges members into a binary tree as above.
- Node1 derives a shared secret between U_0 and U_1 by a Key Encapsulation Mechanism (KEM) that combines their respective public keys pk_0 and pk_1.
- The $shared\_secret$ can be calculated with a PKE such as Diffie-Hellman, etc.
- The $shared\_secret$ is ratcheted with additional metadata (e.g. U_0, U_1) with a KDF

$$\text{Node1} = \text{KDF}(shared\_secret_{0,1} \| U_0 \| U_1)$$

- The same process occurs at Node2 (with U_2, U_3), and at Root (with Node1, Node2).
- All members may communicate with each other using the shared key at root.

Adding a member to the Tree



- a New_Root is added to the tree along with Node3.
- U_4 determines their key pair (sk_4, pk_4).
- Node3 is derived by

$$secret_4 = \mathrm{DH}(pk_4)$$

$$\mathrm{Node3} = \mathrm{KDF}(secret_4 \| U_4)$$

- New_Root is derived by

$$\mathrm{shared\_secret}_{0,\ldots,4} = \mathrm{DH}(shared\_secret_{0,\ldots,3}, secret_4)$$

$$\mathrm{New\_Root} = \mathrm{KDF}(shared\_secret_{0,\ldots,4}) \| \mathrm{Old\_Root} \| \mathrm{Node3})$$

Removing a member from the Tree Say we need to remove U_3 from the above tree.

- The affected subtree nodes would be Old_Root and Node2.
- All nodes along the path to the root of the subtree must be rekeyed.
- Each affected node computes a new $secret/shared\_secret$ which excludes U_3's involvement.

Conflicting Updates Inherently the TreeKEM structure is asynchronous, all members must have the same model of the overall structure, otherwise will result in failure of the encryption scheme (Key State Divergence).

There are several ways of handlign Conflicting Updates. Described in the lecture, upd_i is sent to the server as a proposed update. No user is allowed an update until the server sends back a corresponding ack_i in the next epoch.

The following selection policies may be used:

1. Server picks one upd_i and rejects all other updates.
2. Server delivers all upd_i in the same order. Users execute updates in the received order.
3. Server delivers all upd_i without ordering. Users execute updates based on a policy, e.g. left-most user‘s first.

2.2 Private Function Evaluation

ElGamal Encryption Scheme uses a base group G, with generator g and order q. Encrypts a message $m \in G$.

KeyGen:

1. Sample x from $1, ..., q-1$
2. Set $sk \leftarrow x$ and $pk = h \leftarrow g^x$
3. Output (sk, pk)

Encrypt: *Takes input m, pk*

1. sample s from $0, ..., q-1$
2. $t \leftarrow pk^s$
3. $c_1 = g^s$
4. $c_2 = m \cdot t$
5. Output (c_1, c_2)

Decrypt: *Takes input (c_1, c_2), sk*

1. $s \leftarrow c_1^x$
2. $m = c_2 \cdot s^{-1}$
3. Output m

Exponential ElGamal Message $m \in \mathbb{Z}_p^*$. Uses the same key generation but deviates slightly in encryption and decryption methods.

Encrypt:

- $c_2 = g^m \cdot t$

Decrypt:

- $g^m = c_2 \cdot s^{-1}$
- Output m (calculated from $\log(g^m)/\log(g)$)

Homomorphic Properties of ElGamal and Exp-ElGamal Encrypted ciphertexts using ElGamal can be homomorphically multiplied with each other. While, ciphertexts in Exp-ElGamal can be homomorphically added with each other. Individually, both of these can be classified as Partially Homomorphic Encryption Schemes, because they exhibit partially but not completely, features of Fully Homomorphic Encryption.

Multiplication with ElGamal: Given $sk = x$, $pk = g^x = h$:

- Ciphertexts:
$$c^{(A)} = (c_1^{(A)}, c_2^{(A)}), \quad c^{(B)} = (c_1^{(B)}, c_2^{(B)})$$
- Multiplication of ciphertexts:
$$c = c^{(A)} \boxtimes c^{(B)} = (c_1^{(A)} \times c_1^{(B)}, \quad c_2^{(A)} \times c_2^{(B)})$$
- Decryption:
$$m = \mathrm{Dec}(\mathrm{sk}, c) = m_1 \cdot m_2$$

Addition with Exp-ElGamal: Given $sk = x$, $pk = g^x = h$:

- Ciphertexts:
$$c^{(A)} = (c_1^{(A)}, c_2^{(A)}), \quad c^{(B)} = (c_1^{(B)}, c_2^{(B)})$$
- Addition of ciphertexts:
$$c = c^{(A)} \boxplus c^{(B)} = (c_1^{(A)} \times c_1^{(B)}, \quad c_2^{(A)} \times c_2^{(B)})$$
- Decryption:
$$m = \mathrm{Dec}(\mathrm{sk}, c) = m_1 + m_2$$

Lattices (you will not have to know this explicitly for the exam)

- For linearly independent vectors $(v_1, ..., v_n)$ in $\mathbb{R}^n$

$$\textbf{lattice } \Lambda = a_1 v_1 + ... + a_n v_n | a_i \in \mathbb{Z}$$

- Λ is a discrete additive subgroup of $\mathbb{R}^n$.
- $(v_1, ..., v_n)$ is called a **basis** which generates Λ.
- For vector $v = (v_1, v_2)$ its length $\|v\| = \sqrt{(v_1^2 + v_2^2)}$.

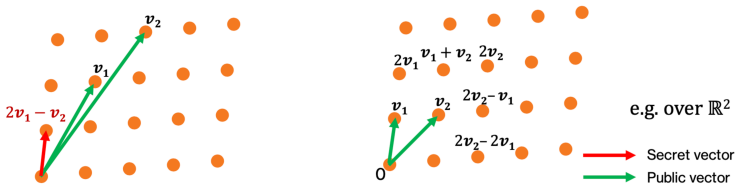


Fig. 4. lattice structure in $\mathbb{R}^2$ space

Shortest Vector Problems in Λ

- **SVP:** Find a short (non-zero) vector s in Λ
- γ**-SVP:** Find a short vector of length $\gamma(n) \cdot \|s\|, \gamma(n) \geq 1$
- **GapSVP**$_\gamma(d)$**:** Distinguish whether $\|s\| \leq d$ or $\|s\| \geq \gamma(n) \cdot d$. (is reducible to a LWE problem)

Learning With Errors (LWE) A computational problem involving a system of noisy linear equations over finite fields. Essentially, given a collection of noisy linear equations, can you recover the original secret that define the equations?

Problem:

- Let $s \in \mathbb{Z}_q^n$ be a secret vector, an odd modulus $q \in \mathbb{N}$
- Oracle $O_{\text{LWE}}(s)$:
  - pick vector $a \in \mathbb{Z}_q^n$
  - picks a small noise term $e \in \mathbb{Z}$ from a noise distribution
  - outputs $b = \langle a, s \rangle + e \mod q$. where;
    - $\langle a, s \rangle$ is the dot product of a and s
    - e is the noise
    - $\mod q$ ensures the result stays within the finite field
- security from difficulty in finding s from polynomially many (a, b)

Peikert Public Key Encryption based on LWE

KeyGen:

1. $sk = s \in_R R_q$
2. $pk = (a, b)$ where $b = as + e$

Enc$(pk, x \in 0, 1)$***:***

1. pick random s', e', e'' from noise distribution X
2. $c_0 = s'a + e'$
3. $c_1 = s'b + e'' + \text{encode}(x)$

$$\text{encode}(x) = x \cdot [q/2]$$

4. return $c = (c_0, c_1)$

Dec(sk, c)***:***

1. $\ddot{x} = c_1 - c_0 s$
2. return decode$(\ddot{x})$

$$\text{decode}(\ddot{\mathbf{x}} \in \mathbb{Z}_q) = \begin{cases} 0 & \text{if } \ddot{\mathbf{x}} \in [-\lfloor q/4 \rfloor, \lfloor q/4 \rfloor) \\ 1 & \text{otherwise} \end{cases}$$

The correctness of Peikert LWE-PKE is since due to $s'b + e'' = s'(as + e) + e'' = s'as + s'e + e'' \approx s'as$, we get $c_1 \approx s'as + \text{encode}(x)$. Simultaneously, since $c_0 s = (s'a + e')s = s'as + e's \approx s'as$. We get $\ddot{x} = c_1 - c_0 s = \text{encode}(x)$.

2.3 Fully Homomorphic Encryption (FHE)

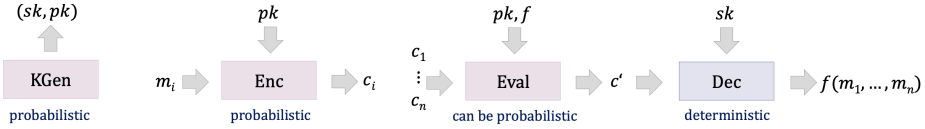


Fig. 5. correctness of FHE scheme

FHE allows the evaluation of arbitrary circuits composed of $(+, \times)$-gates of unbounded depth. HE schemes with bounded depth are defined as **Leveled Fully Homomorphic Encryption**. And partially homomorphic encryption schemes are described above in **Homomorphic Properties of ElGamal and Exp-ElGamal**.

Components of FHE algorithms All FHE schemes have the 3 fundamental components of a PKE scheme and an evaluation function. **Eval**() is where $(\boxplus, \boxtimes)$ actions are applied to the given ciphertexts.

- **KeyGen**(K): generates key pair (sk, pk)
- **Enc**(pk, m): encrypts m with pk and outputs c
- **Dec**(sk, c): Decrypts c using sk and outputs either m or $error$
- **Eval**$(pk, f, c_1, ..., c_n)$: outputs c' that encrypts $f(m_1, ..., m_n)$.

No IND-CCA security in FHE The key feature of FHE is the malleability of ciphertexts. (including ElGamal and all other PHE schemes) By definition, IND-CCA cannot be achieved because the oracle must leak some information which the attacker can use to predict related plaintexts. (e.g., attacker chooses m_0, m_1, challenger encrypts one of them as c, the attacker calculates $(c \boxtimes \text{Enc}(m_0)^{-1})$ and if the result equals 1, the challenger chose m_0, otherwise it is m_1.) [1]

Bootstrapping The key concept for all FHE schemes today require some form of noise being introduced. The issue with this is that when $\boxtimes, \boxplus$ operations are used, the noise can grow to untenable levels (especially with $\boxtimes$). The solution introduced by Gentry in 2009 was Bootstrapping, which would reduce the noise to a fixed manageable level whenever needed.

The idea is performing a decryption within the evaluation function by encrypting the secret key, thereby performing decryption in the ciphertext space. Basic decryption circuits only perform the decryption for the sole purpose of reducing noise. Augmented decryption circuits perform the decryption during homomorphic evaluation $(\boxplus, \boxtimes)$.

Notably, There is still a relationship between sk and pk. However, there is no evidence so far whether this impacts the security of this scheme or not. This circular security may be avoided by using **chain-key switching**, which uses d different key pairs where d is the length of the chain. However, this limits the number of evaluations to $(d-1)$ therefore schemes that utilize this can only be classified as levelled-FHE schemes instead of fully FHE.

Key switching however has the added benefit of extending single-hop SWHE to multi-hop SWHE with $n < d$ hops.

[1] IND-CPA can be achieved because even when the attacker is able to manipulate the contents of ciphertexts, using the attack above without the decryption oracle, he cannot tell whether that ciphertext corresponds to 1 or not.

3 Week 9: Privacy Preserving Machine Learning

The usefulness for protecting privacy of groups and individiduals is obvious. Sensitive, military, medical and financial private data cannot be stored unsecurely and cannot be disclosed with or without discretion. However, with techniques such as;

- Differential Privacy
- Homomorphic Encryption
- Multi-Party Computation
- Federated Learning

Processing and evaluating private data can be done without ever revealing the contents of that data.

3.1 Homomorphic Encryption: CryptoNets FHE

Algorithms

- **KeyGen**(λ) : returns key pair (sk, pk) and bootstrap key bsk
- **Encrypt**(pk, m) : returns ciphertext c
- **Decrypt**(sk, c) : returns message m
- **Eval**$(f, \{c\})$: (potentially many) c, returns $f(\{c\})$
- **Bootstrap**(bsk, c) : takes input 'old' c, key bsk, returns c'

Some Parameters and Variables:

$$\text{Ring } R_t = \mathbb{Z}_t[x]/(x^n + 1)$$

$$\text{Ring } R_q = \mathbb{Z}_q[x]/(x^n + 1)$$

$$\text{polynomials } f', g \in_R R_q^n$$

$$c_1 = ([q/t]m_1 + e_1 + hs_1) \mod q$$

$$c_2 = ([q/t]m_2 + e_2 + hs_2) \mod q$$

Homomorphic Addition where $m_3 = m_1 + m_2$

$$c_1 + c_2 = ([q/t](m_1 + m_2) + (e_1 + e_2) + h(s_1 + s_2)) \mod q$$

$$= ([q/t]m_3 + e_3 + hs_3) \mod q$$

This has the same shape as a normal ciphertext and will decrypt to $m_1 + m_2$, provided e_3 is small enough.

Homomorphic Multiplication where $m_3 = m_1 \times m_2$

$$(q/t)(c_1 \times c_2) = ([q/t](m_1 m_2) + (e_1 e_2) + h^2(s_1 s_2)) \mod q$$

$$= ([q/t]m_1 m_2 + e' + h^2 s_1 s_2) \mod q$$

This has the same shape as a normal ciphertext and will decrypt to $m_1 m_2$ if e' is small enough. The secret key must match the shape of the public key h here so is now **squared**. $(sk = f^2)$

Scalar Operations Not all operations require encryption. e.g., weights in NNs are known to the network, therefore there's no need for their encryption.

Scalar Addition: *with weight* w

$$c + [q/t]w = ([q/t](w + m) + e + hs) \mod q$$

Essentially just encrypting w with no noise and performing normal homomorphic addition.

Scalar Multiplication: *with weight* w

$$c \times w = ([q/t]wm + e' + hs') \mod q$$

This is very efficient, especially if w is sparse.

Practical Considerations The maximum noise that a ciphertext can have and still be decryptable depends on q and t. Therefore q should be selected to be large enough to accommodate the increasing noise, which then necessitates choosing a larger n for security reasons.

Keeping parameters small may improve performance for these tasks, however we would like to make t large to prevent coefficients of the plaintext polynomials from reducing $\mod t$ at any point during the computation. This is especially problematic for NNs where we have to encode inputs.

Parameter Selection and Amortisation Parameters t_1, t_2 are plaintext moduli chosen so their product is greater than 2^{80} (large enough for most NNs). But small enough so that coefficient modulus $q = 2^{383} - 2^{33} + 1$ and polynomial modulus $(x^n + 1) = x^{8192} + 1$ allow so that the noise does not grow too large. The plaintext moduli are chosen such that the polynomial modulus breaks into linear components which allow for optimal use of SIMD (Allowing the same message to contain 8192 images instead of just 1). Which poses a different challenge as now it requires data to fit neatly into batches of n (8192 images).

The Activation Function FHE can only compute Addition and Multiplication. In the CryptoNets scheme there are no ways of FHE to compute non-linear functions, such as most activation functions. Therefore a new activation function that can be modelled from $\boxplus, \boxtimes$ functions.

The Square Activation Function:

$$f(x) = x^2$$

This function exhibits strange behaviour in that, unlike ReLU and Sigmoid, the derivative of the Square is that its derivative is unbounded and can make weights 'blow up' or overfit.

In CryptoNets, the final layer must still be a Sigmoid Activation Function to achieve reasonable error terms when gradient descent is applied. However because the sigmoid function is monotone increasing, this can be left out of training after all weights have optimised, where it can be done in plaintext space.

3.2 Torus FHE

Algorithms

- **KeyGen**(λ) : returns key pair (sk, pk) and bootstrap key bsk
- **Encrypt**(pk, m) : returns ciphertext c
- **Decrypt**(sk, c) : returns message m
- **Eval**$(f, \{c\})$: (potentially many) c, returns $f(\{c\})$
- **ProgBootstrap**(bsk, c, g) : takes input 'old' c, key bsk, an optional small-function g, returns c'

Programmable Bootstrapping As with bootstrapping, PBS resets the noise value to a manageable state by decrypting the ciphertext in the encrypted domain.

Simultaneously, it allows the ability to compute an arbitrary function g over the ciphertext c, provided it can be encoded as a small lookup table. Meaning, function such as ReLU and Sigmoid can be directly implemented within the FHE framework, over a small domain as part of bootstrapping.

PBS introduces a new global parameter that controls exactness. With probability p, the PBS lookup will return the wrong entry from the lookup table.

$$\text{smaller } p \Rightarrow \text{larger parameters} \Rightarrow \text{less efficient}$$

The error distribution is normal, centred on the correct value. So if it returns a wrong answer, it probably isn't too far away.

Encoding Problems ML models and algorithms typically use floating-point arithmetic to approximate real numbers. TFHE can only work over integers. A method is needed to translate these values.

Quantisation Quantisation is the process of constraining an input from a continuous or otherwise large set of values (such as real numbers) to a discrete set (such as integers).

Some accuracy in representation is lost as usually the least-significant bits are eliminated. In many cases, for ML it is possible to adapt the models to give meaningful results while using these smaller data types. Which significantly reduces the number of bits necessary for intermediary results during execution. (TFHE currently is limited to 16-bit integers)

Quantising $\mathbb{R}$ *to* $\mathbb{Z}$ *space.* Let $[\alpha, \beta]$ be the range to quantise, $n = 8$ be the number of bit integers.

$$S = \frac{\beta - \alpha}{2^n - 1} = \frac{\beta - \alpha}{2^8 - 1} = \frac{\beta - \alpha}{255}$$

S is the scale parameterised by n. The range is $[0, 255]$.

To make quantisation computations hardware-friendly, ensure that scales are in powers-of-two. This allows division and multiplication by 2 to be computed by shift operations.

Quantisation Aware Training (QAT) Let n be the bit-width of the inputs, and *nbits* denote the bit-width of the weights.

Linear models only do quantisation post-training. The model is trained in floating point, then the best integer weight representations are found, depending on the distribution of inputs and weights.

Tree-based models need training and test data to both be quantised. the value of *nbits* bits is known beforehand ($nbits = n + 1$ bits).

For NNs, the bit-width cannot be precisely controlled. For better model accuracy, it is beneficial to have many input features and a high number of bits. But this drastically hinders performance, a desirable compromise in both can only be found through experimentation.

4 Week 10: Privacy Preserving Machine Learning cont.

Turning training data into a good model is not necessarily one-way. How does the model come to the correct conclusion about new and unseen data? Clearly, in order to be able to identify and map features to labels, the model needs to extract information from training.

Therefore, this probably does pose some form of privacy issue. Because it allows someone with access to the model to deduce information about the training data.

4.1 Training Privacy Attacks

Querying the model inherently leaks knowledge about the training set. The problem is how can you distinguish between honest and adversarial queries?

Membership Inference Models tend to perform better on data they are trained on. Given a set that includes data used in training, an adversary is able to infer which examples were used in training by their performance.

The adversary does not even need to have knowledge of about the parameters of the model. Just the algorithm and architecture.

Model Inversion The ability of an attacker to reconstruct the input data used to train the model. This also requires the use of querying.

Fig. 6. left: reconstruction from adversarial queries, right: original training datum

4.2 Differential Privacy

To protect against these attacks, two approaches can be utilised. First, adding noise to the outputs. Or second, adding noise to the inputs (and/or to the model)

Adding Moise Blurs the true value of training updates. This should be done carefully so that the noise is small enough that the parameters still remain approximately correct. But should be large enough that reverse engineering data is not possible.

One Concern: Sure, it is differentially private for 1 update for 1 data sample. But training a model uses many data samples and many updates.

Amplification Theorem For a dataset $|D|$ with sample size $|L|$, define $\frac{|D|}{|L|}$.

Theorem 1 (Amplification Theorem). *If an update $\triangle\theta$ is (ε, δ)-Differential Private within a sample, then it is $(O(q\varepsilon), q\delta)$-Differential Private within the dataset.*

So one update for all data is differentially private.

Composition Theorem

Theorem 2 (Composition Theorem). *Applying a $(\varepsilon_1, \delta_1)$-Differential Private algorithm, then a $(\varepsilon_2, \delta_2)$-Differential Private algorithm, gives a $(\varepsilon_1+\varepsilon_2, \delta_1+\delta_2)$-Differential Private algorithm.*

Many updates for all data is differentially private.

Theorem 3 (Strong Composition Theorem). *Applying the same (ε, δ)-Differential Private algorithm T times gives a Differential Private algorithm with parameters:*

$$(O(\varepsilon\sqrt{Tln(1/\delta)}), T\delta)$$

4.3 Federated Learning

A machine learning setting where multiple entities (clients) collaborate in solving a machine learning problem, under the coordination of a central server or service provider. Each client's raw data is stored locally and not exchanged or transferred; instead focused updates intended for immediate aggregation are used to achieve the learning objective.

Security Concerns:

1. Data must remain private to the organisation that owns it.
2. Individual Models remain private to the organisation that created it.
3. The combined model does not break 1 or 2.

Averaging Federated averaging averages the results of models from each client. The clients each perform training and have a local dataset. The server responsible for aggregating weights does not necessarily needed to be powerful.

The model is first initialized on the server. At each round t, a random set of clients are chosen and each client performs local gradient descent steps, and the server aggregates model parameters submitted by the clients.

Cross-Device This FL process involves large number of devices, such as smartphones and IoT devices that participate in training a global model collaboratively.

The scale of CDFL can be much larger than CSFL, participants do not necessarily have large computational power and do not need to contribute large amounts of data.

Data is unique to each device and varies significantly between participants (Highly Non-IID).

Cross-Silo CSFL involves smaller number of organisations (silos). This is used for medical, financial or corporate institutions that collaborate to produce global models while keeping their data private.

The scale is usually much smaller than CDFL and usually participants require substantial computational resources as they need to compute their own large datasets. Data distribution is usually moderately Non-IID.

Challenges of FL

- Data Heterogeneity: Data generated by each user can be very different (format, type, etc.)
- Unbalanced Data: Some users produce more data than others
- Limited Communication: Unstable Mobile Networks (affects CSFL more)
- Hardware Heterogeneity: Potentially extreme ranges of compute and memory of devices
- Privacy, Security and Trust: Malicious Clients and Servers

4.4 Secure Aggregation

Consider that every participant is an adversary, and that their goal is to learn everything about every other participants data (such as financial or capital firms). Secure Aggregation is the concept of ensuring the privacy of participants data while enabling the computation of a global model.

Federated Learning Participants apply techniques such as adding noise to their data and model, while the server, when summing each participants models, simultaneously applies a mask. When all parts are aggregated, the sum of the masks will cancel out.

This all assumes however that the server is to be trusted. What if that were not the case?

Homomorphic Encryption Say client c_1 and client c_2 want to aggregate their models. c_1 encrypts their model as up_1 with a Homomorphic PKE scheme and sends that to c_2. c_2 does the same, creates up_2 and adds it to up_1. The server can only learn $up_1 + up_2$ but not know their contents individually.

- Even against a honest but curious server, the client models remain IND-CPA secure.
- Sequential ordering means any drop out invalidates the entire process
- No way of parallelisation
- A single malicious client can invalidate the whole sequence, and in tandem with the server may break privacy of other models.

Multi Party Computation Devices cooperate to sample random vector pairs of 0-sum perturbation vectors, which cancel out during aggregation.

Each user computes a secret update vector w_i they want to contribute to the global model. Each pair (i, j) needs to compute a shared secret $s_{i,j}$. Denote $s^+_{i,j} = s_{i,j}$ and $s^-_{i,j} = -s_{i,j}$

Each party computes and shares the following:

$$c_i = w_i + \sum_{j \neq i} s_{i,j} \mod p$$

Finally the server computes:

$$\sum c_i \mod p = \sum w_i + \sum_{j \neq i} s_{i,j} \mod p = \sum w_i + \sum_i (s^+_{i,j} + s^-_{i,j}) = \sum w_i$$

- Even against a malicious server, the client models remain private.
- All parties must participate, may not be useful for CDFL where availability may not be assumed
- No protection against a malicious client
- Computationally friendly compared to HE approach. (only addition calculations)