Homomorphic encryption & encrypted data processing

COSC312 / COSC412

Learning objectives

- Describe some types of **useful work that can be done on encrypted data**, and risks in encrypting storage
- Appreciate the overall way in which an example **homomorphic encryption** scheme operates
- Understand the potential usefulness of homomorphic encryption in the context of **cloud computing**

Non-malleability

- Attacker usually shouldn't be able to make any controlled **changes to deciphered data**
- This can be a property of the cypher in use …
  - *e.g.*, as seen previously: many block cipher modes
- … or a property of how the cypher is used
  - *e.g.*, ensure that there is a checksum in the data that has been encrypted
  - thus **tampering is noticed** even if decryption did not fail

Malleability

- Errors in stream ciphers showed **malleability**
  - If attacker can introduce a cipher-text bit error, there's a change in the corresponding, **decoded plain-text bit**
- More mathematically: $[m]_k = m \oplus S(k)$
  - Where: m—message, k—key, $S(k)$—key stream, $\oplus$—XOR
- Attacker generates $[m]_k \oplus n$ (where n is attack string)
- $[m]_k \oplus n = m \oplus S(k) \oplus n = (m \oplus n) \oplus S(k) = [m \oplus n]_k$

- Attack requires victim **not to detect change**

Homomorphic encryption

- It is possible to perform useful computations on data by manipulating cypher-text
- Apply **malleability for good** (it's usually undesirable)
- Two broad classes of homomorphic cryptography
  - **Partially Homomorphic Encryption** (PHE)
    - Some existing ciphers already provided PHE reasonably efficiently
  - **Fully Homomorphic Encryption** (FHE)
    - Old schemes expensive; newer schemes orders of magnitude faster

Partially Homomorphic Encryption

- One type of operation can be computed
  - e.g., can compute encrypted sum of two encrypted values without decryption

- Specifically for **Paillier**:
  - With public key k, and encrypted messages $[m_1]_k$ & $[m_2]_k$
  - Can compute $[m_1 + m_2]_k$ by multiplying $[m_1]_k$ and $[m_2]_k$
- For **ElGamal** & **RSA**:
  - With $[m_1]_k$ and $[m_2]_k$
  - Can form $[m_1 \times m_2]_k$ by multiplying $[m_1]_k$ and $[m_2]_k$

Somewhat Homomorphic Encryption

- FHE has complete ring structure (two gate types), thus:
  - **general code can be translated** (at least in theory)
    - FYI: logic circuit can be encoded as polynomials on algebraic ring
  - internal state is not disclosive
- Helpful: **somewhat homomorphic encryption** (SHE)
  - Can do two gate types but for circuits with limits
  - Craig Gentry's first scheme (2009) **bootstraps SHE into FHE**:
    - essentially "refresh" encrypted data using homomorphic approach
    - avoids encrypted data vanishing into noise during computation

Fully homomorphic encryption

- Since Gentry's first approach in 2009, much progress:
  - Multiple generations of improved schemes
  - Many open source implementations available

- Different implementations suit different use cases
  - e.g., recent schemes support some types of machine learning

- **Homomorphic Encryption Standardisation** in 2017
  - Includes NIST, IBM, Microsoft and Intel

Cloud computing

- **Outsourcing of computation and storage**—benefits:
  - Avoid fixed costs of infrastructure
  - Best practice in persistence and management
  - Geographically spread (potentially)
  - Elastic—can scale up on demand
- A key **downside—security**: gaining trust, privacy, *etc.*
  - The cloud provider is not your organisation
  - Further problems arise when crossing jurisdictions
    - *e.g.*, EU General Data Protection Regulation (GDPR); US CLOUD Act

Homomorphic encryption + cloud

- Can facilitate same outsourcing as before … but without cloud provider seeing raw data
  - Cloud providers can still deny service
  - … but clients can compensate: use multiple cloud providers
- Cloud can support some inefficiency through elasticity
  - … but not too much or it becomes uneconomical
  - Potential utility would justify FHE hardware accelerators
- September 2017: Azure confidential computing (SGX)

Homomorphic encryption for AAA

- One potential focus for homomorphic encryption and cloud computing is in AAA (auth, authz & accounting)
  - As seen previously, authentication and authorisation often involve small amounts of data processing
- In normal operation, homomorphic access control policy evaluator can't see policy meaning or state
  - Thus can have a third party trusted organisation that manages key escrow (akin to Kerberos' KDC)
  - This way, can build a **distributed access control system**

Doing useful work on encrypted data

- Encryption effects confidentiality
  - Third parties can transport encrypted data
    - *e.g.*, in the sense of networks, or of storage systems
  - Third parties cannot usefully modify encrypted data
    - Doing so will destroy the data, usually detectably
- However ideally those third parties touching confidential data can do useful work on it
  - … although clearly confidentiality must remain
  - Those parties could always have denied service (availability)

Useful work on encrypted data

- Encrypted data can be **structured** so that useful work can be done without decrypting it
  - Note that these approaches **do not modify the chunks** of encrypted data

- This is in contrast to homomorphic encryption
  - The encrypted data is modified under homomorphic crypto.
  - However the data being modified remains confidential: doing modification does not imply being able to decrypt the data

Encrypted search

- Seen previously: if encrypted data is not salted, in many schemes, if $a = b$ then $[a]_k = [b]_k$
  - *e.g.*, Adobe password database disclosures provided unsalted encrypted passwords, and unencrypted password hints
- This property can be useful: *e.g.*, **search done by a third party** from whom data is hidden
- Recall mention of **structure** in such schemes
  - Consider implications for key-value storage

Encrypted search

- **Separately encrypt** the key and value
  - High capacity key/value storage and/or database engines can efficiently index encrypted values on encrypted keys
- Schemes have **extended SQL interfaces** to facilitate this type of encrypted search
- Most straightforward approaches are limited to performing **equality** testing
  - No support for range queries

Filtering rather than searching

- Rather than finding a particular key, instead **use encrypted attribute to cluster data**
  - *i*.e., expect that an encrypted search will return many records
- Subsequent filtering can occur at the client
  - Get useful filtering: *i*.e., large-scale database helps the client to not look at records that are determined to be irrelevant
- We can extend this idea to use multiple attributes

Some support for range queries?

- So let's assume we have an **ordered key**
  - Client knows all the bits in the key
  - We can group bits into **separate encrypted attributes**

- A given record can be retrieved by requesting **disjunction of encrypted attributes** from the database:
  - *i.e.*, as a set of independent equality tests on encrypted data
  - database does not get to know the bits…
  - however there are risks of revealing correlations

Query trees for range queries (1)

- A range query can be expressed as a **set of equality tests** on constituent bits of key
- With 4 bits, express retrieval of elements less than 5:
  - 5 is 0101_2, so we want:
  - 0100 and 00?? (where ? is any bit)
  - *i.e.*, **just two queries** (in this case)
- Easily extended to more complex inequalities
  - Also, no requirement to have single-bit-level encryption

Query trees for range queries (2)

- Get expensive query expressions, but they still perform a **useful filtering role** quickly—utility depends on queries
- Risks: database potentially learns a lot
  - Can try to counter this by **adding noise**
    - *e.g.*, make additional queries for data that you don't actually want
    - … but then the noise needs to be effective
    - Access statistics may allow a malicious database to filter noise
  - Alternatively, **use redundancy in coding** of bit patterns
    - *i.e.*, provide multiple different ways to filter out the same dataset

A useful cloud service: managed storage

- External storage: large economies of scale
  - De-duplication of shared data
  - Defragmentation of free space
  - Multi-tier storage systems
    - RAM; SSD; spinning disk; tape
- **Problem**: many staff need to access data: *e.g.*,
  - sysadmins **monitoring infrastructure**
  - operators **generating external backups**

Seeing encrypted data: key escrow

- Cloud storage: usually **encrypts data at rest**
  - The third party can still block availability to the client
  - Ideally we want a system that **encrypts data at the client-side**
    - But adds the usual difficulties of managing client-side software
- Encrypted data is a double-edged sword
  - Underlying storage media can block availability
- Key escrow: **key ownership is shared**
  - but … obligation to give up keys to authorities?

Groups and key management

- Key escrow: group of principals can decrypt
  - One approach: extend cryptographic methods
- Far easier: use a **multi-stage cryptography process**
  - Encrypt data with one-time symmetric key k_s
  - Use asymmetric cryptography to encrypt k_s for each principal
- Can also **require collaboration to decrypt**
  - Threshold number of keys must be presented
  - Organisations must agree on the need for disclosure

Building reliable (available) storage systems

- Need to **ensure updates are crash-safe**
  - Journaling added to conventional filesystems
    - e.g., NTFS, HPFS+, Ext3
  - Entire **copy-on-write filesystems**
    - e.g., ZFS, BTRFS, ReFS, APFS, WAFL (NetApp)

- Replication, e.g., **RAID schemes**
  - Can handle some number of devices going offline
  - But what about handling corrupted data?

Encrypted filesystems need reliable storage

- Mentioned previously: many **OSs offer encrypted FSs**
  - … although notably TrueCrypt died without much explanation:
    - a particular pity given that **TrueCrypt did steganography**
- Really want filesystem to actually **verify your data**
  - Otherwise bit errors will most likely cause data loss
- ZFS, ReFS, or BTRFS (but not APFS!) can ‘scrub’ disks
  - Combined with RAID, can keep encrypted data safe

Repacking encrypted data

- Another use for encrypted, structured data

- Useful to package files into archives
  - Particular use case: HTTPS upload
  - Used not to handle multiple files effectively
  - Instead pack files into a ZIP and upload that

- What if the data is sensitive?
  - Can employ ZIP files that use a password

How encrypted ZIP files work

- Before considering how to repack them, need to know what we are repacking
  - Why are TAR.GZ files (often) smaller than ZIP?
  - How do self-extracting archives work?
- ZIP: **each file is stored in a chunk**
  - There's also a **table of contents** in order to collect metadata
- Encryption protects data, not metadata
  - Sometimes the filenames may be sensitive (use 7zip instead…)

Repacking encrypted ZIP content

- Needed simplicity of HTTP upload using ZIPs
  - Allows easy **upload of large encrypted data files**
- Want users to be able to **download subsets**
  - Research project had n-to-m interactions
- Thus can treat compressed files as opaque
  - instead **reorganise blocks into subsets**
  - regenerate the metadata for the new archive

In summary

- Introduced **homomorphic encryption**
  - Differentiated PHE and FHE schemes
  - Gave a sketch of the operations possible
- Provided an overview of **cloud computing** and how it can make use of the **above techniques**
- Discussed useful operations that third parties can **perform on encrypted data**: (and some storage risks)
  - *e.g.*, storage, data repacking, and search