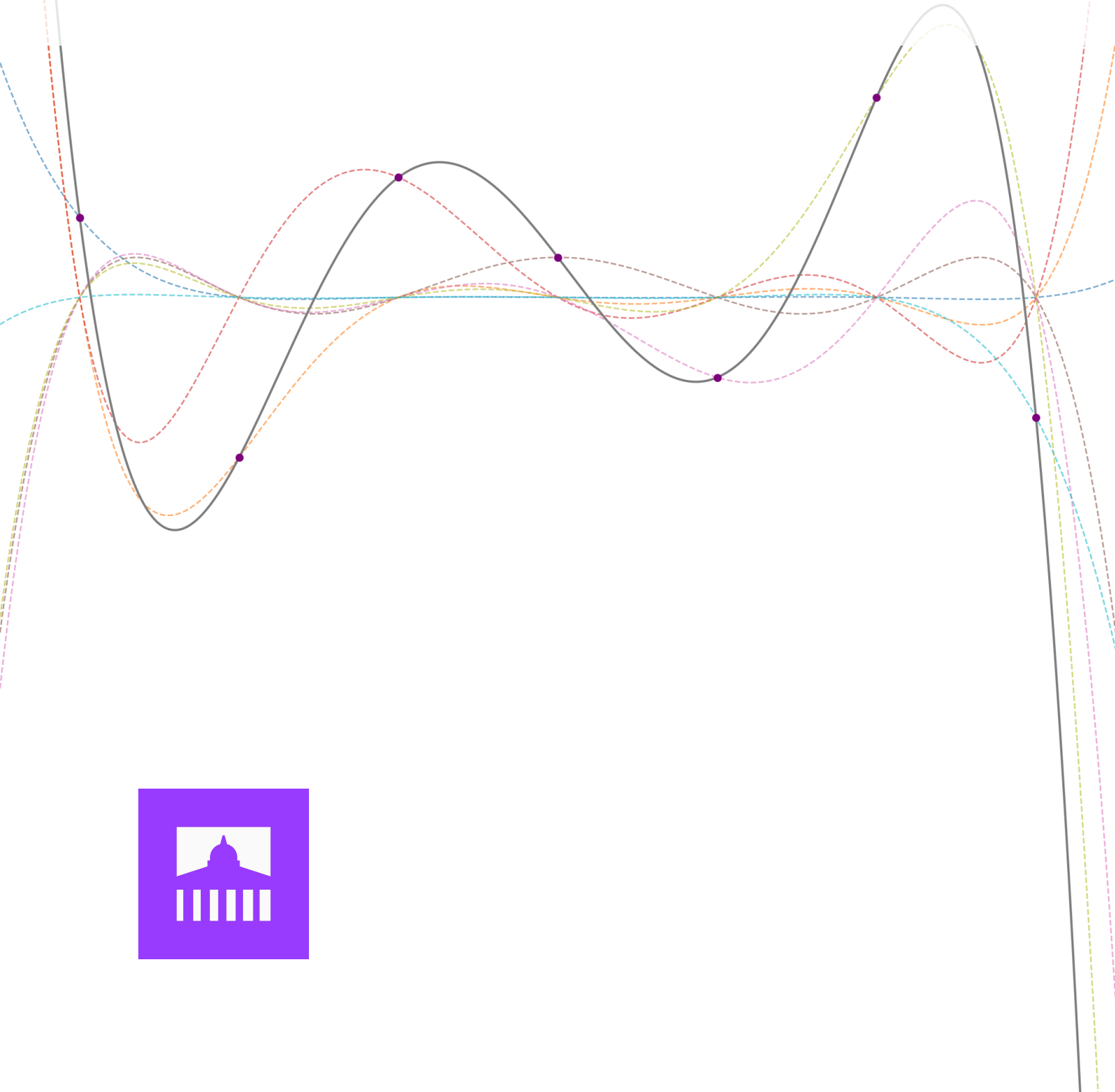
COMP0061: Privacy Enhancing Technologies

Pascal U

Lecture Notes, Spring 2026



1 Solove's taxonomy of privacy

Before starting anything, we begin with Solove's taxonomy of privacy.

- **Information collection** can cause harm in itself
  - e.g, surveillance, interrogation
- **Information processing** can lead to new harms
  - e.g, aggregation, identification, insecurity, secondary use, exclusion
- **Information dissemination** - Sharing information inappropriately
  - e.g, breach of confidentiality, disclosure, exposure, increased accessibility, blackmail, appropriation, distortion
- **Invasion** - Disrupts someones activities
  - e.g, intrusion, decisional interference

Solove gives examples of harms that can result from violation of privacy, For example, a newspaper reports the name of a rape victim, a reporter deceitfully gains entry to a person's home and secretly photograph the person, the government uses a thermal sensor device to detect heat patterns in a person's home, a company markets a list of five million elderly incontinent women.

2 Hard and Soft PETs

Soft PETs

- Focus on compliance
- Focus on "internal controls"
- **Assumption**: a third party is entrusted with the user data
- **Threat model**: third party is trusted to process user data according to user wishes
- e.g., access control, tunnel encryption (SSL/TLS)
- "Keeping honest services safe from insiders/employees"

Hard PETs

- Focus on data minimization
- **Assumption**: no single third party may be trusted with data (e.g., k out of n honest parties)
- **Threat model**: a service is in the hands of the adversary; may be coerced; may be hacked
- May rely on service integrity if auditing is possible
- **Challenge**: achieve functionality without revealing data

PETs must protect confidentiality of sensitive information from adversaries but how do we do so without revealing information to third parties?

- **Peer-to-peer/ End-to-end**: no third parties, just do everything amongst ourselves
- **Dumb relays/ Dumb stores**: use third parties, but they cannot see any information, likely by encrypted communication/storage.
- k**-out-of-**n: we use third parties, but without allowing any coalition to learn anything, e.g, secure computation.

Communications Privacy

Privacy Enhancing Technologies (COMP0061)

Steven Murdoch | Sem 2 25/26 | 12-16 Jan

---

3 End-to-End (symmetric) encryption

Modern cryptography provides **strong** and **surprising** guarantees (if well implemented). When Alice and Bob share a secret key that is difficult to guess, they may encrypt messages that only those with the key can decrypt **(confidentiality)**, and ensure that the messages decoded were encoded by someone who knew the correct key **(integrity & authenticity)**.

Asymmetric cryptography allows Alice and Bob to communicate privately over public channels without a shared secret. A public key is a mathematical object which can be used in combination with a private key to encrypt messages.

- **Key Derivation:** Using their respective public and private keys, both Alice and Bob are able to derive a symmetric key known only to them.
- **Public Key Encryption:** Alice may encrypt a message that only Bob can decrypt.
- **Signing:** Only Alice may sign a message, which anyone can verify the signer knows Alice's private key.

Modern symmetric cryptography carefully combines authentication with encryption. Suppose Alice and Bob share a key K. Where each bit in K must be indistinguishable from random to any third party (adversary).

3.1 Authenticated Encryption with Associated Data

- **Authenticated:** An adversary may not change any public or confidential part of the message without the decryption failing.
- **Encryption**: an adversary cannot learn "anything" about the message or the key by observing ciphertexts or plaintexts. Different ciphertexts of the same plaintexts look different (and cannot be linked).
- **Associated data**: protects the authenticity of some non-encrypted data.

$$\text{Alice} \xrightarrow{C=E_K(P)} \text{Bob}$$

We may use NISTs Advanced Encryption Standard (AES) block cipher in Galois Counter Mode (GCM) but we have to ensure that it's correctly implemented.

- Key lengths: 128, 192, 256 bits
- Initialization Vector: must be fresh random for each encryption (no reuse)
- Tags: (of block length) should be generated for each encryption.
- Assoc. data: ciphertexts must be same bit length as plaintext.

$$\text{Alice} \xrightarrow{<\text{IV,AD,(Tag,C)}=\text{AEAD}_k(\text{AD,P})>} \text{Bob}$$

4 End-to-End (asymmetric) encryption

Public key cryptography eases the problem of distributing symmetric keys. If Alice and Bob have never met before, how can they share a key? The Diffie-Hellman protocol allows the ability to establish a symmetric key over a public channel.

Consider group $\mathbb{G}$ with elements $g \in \mathbb{G}$ and $h \in \mathbb{G}$.

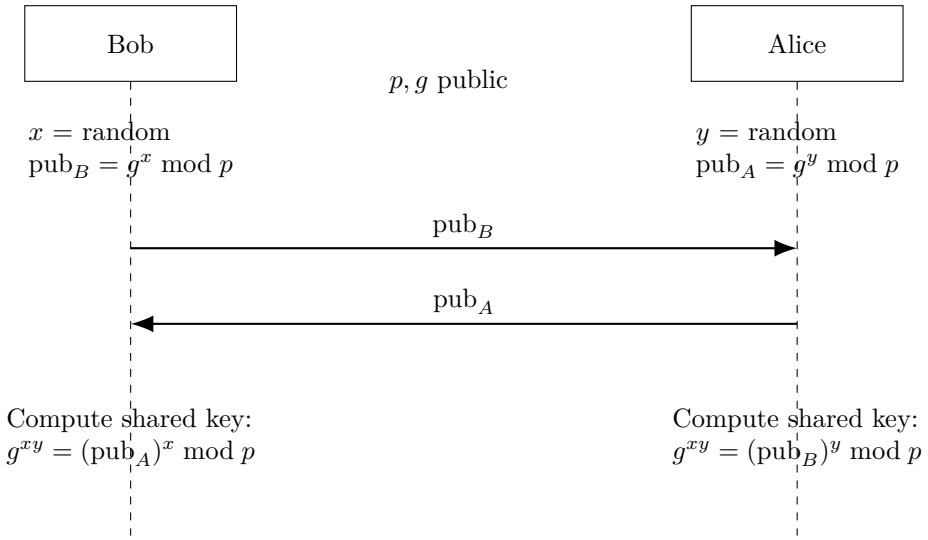
- e.g, 1. group integers $x \mod$ 1024-bit prime.
- e.g, 2. group pairs of integers $(x, y) \mod$ 256-bit prime satisfying EC equation $y^2 \equiv x^3 + ax + b \mod p$.

Two aspects which define group $\mathbb{G}$ are the commutative operation $\oplus$ between elements, and a "0" element `s.t.` for any $g' \in \mathbb{G}$ we have that $g' \oplus 0 = g'$

4.1 Hardness Assumption

For a scalar integer x, it is easy to apply $\oplus$ x-times on an element $g \in \mathbb{G}$. This is denoted by $x \otimes g$. Given $g \in \mathbb{G}$ and $x \otimes g$, it is hard to determine x. e.g, Let p be a large prime, given $g \in [1, p-1]$ (and x), it is easy to determine $\text{pub} = g^x \mod p$, but given g and pub, it is hard to determine x (discrete logarithm problem). We can therefore use this property to establish a private symmetric key using a key exchange protocol.

4.2 Diffie-Hellman Key Exchange



4.3 Elliptic Curves defined over $\mathbb{R}^2$

For integers modulo a prime, p must be 1024 to 2048 bits to be secure. These are computationally slow, so elliptic curves are increasingly more popular as they can be secure under smaller primes.

Group elements (x, y) `s.t.` $y^2 \equiv x^3 + ax + b \mod p$. We define addition given a line between two points P_1, P_2, find the intersection and reflect.

- **We define the neutral element as "inf", the "point at infinity":**
  - It cannot be represented as (x, y) for $x, y \in \mathbb{Z}_p$.
  - $\text{inf} \oplus (x, y) = (x, y)$ and $\text{inf} \oplus \text{inf} = \text{inf}$.
- **Define point addition:**

$$(x_r, y_r) = (x_p, y_p) \oplus (x_q, y_q)$$

`s.t.`

$$\lambda = (y_q - y_p)(x_q - x_p)^{-1} \pmod p$$
$$x_r = \lambda^2 - x_p - x_q \pmod p$$
$$y_r = \lambda(x_p - x_r) - y_p \pmod p$$

- **Special case, point doubling:**

$$(x_r, y_r) = (x_p, y_p) \oplus (x_p, y_p)$$
$$\lambda = (3x_p^2 + a)(2y_p)^{-1} \pmod p$$
$$x_r = \lambda^2 - 2x_p \pmod p$$
$$y_r = \lambda(x_p - x_r) - y_p \pmod p$$

Elliptic Curve point scalar multiplication $x \otimes g$ can be naively implemented as applying $\oplus x$ times to g. A better method is double-and-add.

- Consider that x is in binary form.
- We initialize an accumulator Q to inf.
- For each bit i, set to one add $2^i \otimes g$ to an accumulator Q.

This method needs only up to $2 \log_2 x$ point additions, and is dependent on the number of 1-bits. Another method is the Montgomery ladder algorithm, where the number of operations are independent on bit representation of x.

5 Hybrid Encryption

Asymmetric cryptography is slow, but is a much more effective form of key management than symmetric cryptography. We can have the best of both by using public keys (asymmetric) to derive shared keys to be used in symmetric cryptographic communication. Further, with the use of digital signatures, we can ensure authenticity of keys.

5.1 long term keys

Alice shares a public key signed using a long term key. For Bob to send Alice a message, he generates a fresh public key (and may sign it using a long term key). using the private part of the key and Alice's public key, Bob derives a shared symmetric key. Bob uses the shared key to encrypt messages using AEAD, and sends the fresh public key and the output of the AEAD.

When Alice receives the message, she uses the public key received and her private key to recover the shared key, then decrypts and authenticates the message.

$$\text{Bob} \xrightarrow{\text{pub}_A, \text{sign}_A(\text{pub}_A)} \text{Alice}$$

$$\text{Bob} \xleftarrow{\text{pub}_B, \text{sign}_B(\text{pub}_B), \ldots, \text{AEAD}_K(P)} \text{Alice}$$

- What happens if no signatures are used?
  - If no signatures are used, then there is no guarantee that the original request was made by Bob.
- Are you sure the designed scheme is resistant to a man-in-the middle?
  - Alice and Bob never reveal their private keys, and due to the security of DLP, an MitM has no method for deriving the shared key.
- What happens if either Alice or Bob are forced to reveal their secrets?
  - read next section.

5.2 Perfect Forward Secrecy

PFS aims to protect content even when key material is leaked, as after a certain point, no information can be coerced into revealing any secrets that decrypt the ciphertext.

For signed fresh public keys:

- We can be sure that coercion to reveal keys is ineffective.
- This is because a sender can delete private keys as soon as a message is encrypted.
- Further, a receiver may delete the secret as soon as the message is received.
- Signatures ensure man-in-the-middle attacks are detected.

Anonymous Communications

Privacy Enhancing Technologies (COMP0061)

Steven Murdoch | Sem 2 25/26 | 19-23 Jan

---

6 Introduction

Network identity today offers neither privacy nor authenticity/integrity. IP and MAC addresses have weak identifiers, location and application data is easily leaked. Weak identifiers are easy to spoof, with authentication only being a partial solution as DoS and network level attacks still get through.

In fact the Ethernet packet format is not private at all with invasive fields shown in plaintext. The IP packet has the same problem, as does TCP, SMTP, IRC, HTTP,....

Anonymity is a useful feature for various communication scenarios.

- Electronic Voting
- Auctions/ bidding/ stock market
- Incident reporting
- Witness protection
- Freedom of speech protections
- Discrimination protections
- Investigation/ market research

6.1 Properties of Anonymity

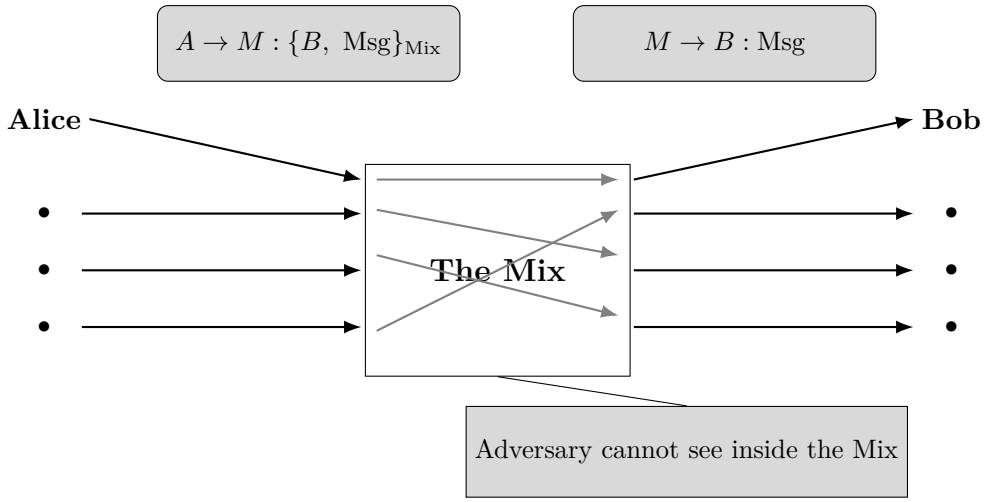
- **Unobservability**: Alice and Bob communicate, but no one can tell if they are transmitting or receiving messages
- **Unlinkability**: Two messages sent (received) by Alice (Bob) cannot be linked to the same sender (receiver)
- **Pseudonymity**: All actions are linkable to a pseudonym, which is unlinkable to a principal

7 High-Latency Anonymity Systems

7.1 Mix networks

First conceptualised by David Chaum in 1979, mixnets make use of cryptographic relays which break the link between sender and receiver.

Essentially, the mix hides the correspondence between input and output messages.



For security, the mix needs to resist traffic analysis and provide bitwise unlinkability. Additionaly, mixes require properties which go beyond just E2EE.

- **Bitwise unlinkability**: ensures adversaries cannot link messages from their bit pattern.
- **Traffic analysis resistance**: messages in and out of the mix cannot be linked to using any meta-data (timing, etc.)

7.2 Basic Attacks

1. An example of **an active attack** against a mixnet; an adversary performs a MitM and intercept a message m from alice and tags it .Then, when the mix outputs m, the attacker can link the message to the receiver, breaking bitwise unlinkability.
2. **A passive attack**; a mix that does not scramble the timing and order of messages is susceptible to traffic analysis. The adversary simply counts the number of messages and assigns to each input the corresponding output.

mitigation strategies:

- Anyone can send messages to the mix, an algorithm is applied and outputs the result, including the attacker - decryption oracle, routing oracle, etc.
- Wait for N messages and output them in random order.
- Pool n messages, wait for N inputs, output N out of $N + n$ and keep remaining n in pool.

8 Mix System Topology

A single mix limits scalability and centralises trust, using multiple mixes distributes the trust and computation load over the multiple mixes. Two ways of implementing this:

- **Mix cascades** - all messages are routed through a preset mix sequence, this is good for anonymity but can have poor load balancing. Here, encryption is nested to prevent an adversary tracing messages through the network.
- **Free routing** - where each message is routed through a random sequence of mixes and relies on the length of the sequence for security. Free route networks set requirements on the cryptography used for messages.

The length of paths through a free routed mix network should be roughly

$$O(\log(|\text{Mix}|)) \text{ steps} = \log(|\text{Mix}|) \text{ cost}$$

while mix cascades have a fixed length path

$$O(|\text{mix}|)$$

8.1 $n-1$ Attack

An active attack against a mixnet where, apart from Alice's message, all incoming messages (trickle) are blocked and the attacker injects their own messages (flood) until Alice's message is out.

Alternative mix designs can mitigate $n-1$. Some designs include;

- **Strong identification**: to ensure distinct identities.
- **Message expiry**: where messages are discarded after a deadline. Preventing adversary from flushing the mix, or injecting messages unnoticed.
- **Heartbeat traffic**: Mixes route messages in a loop back to themselves, able to detect whether an adversary is blocking messages, which forces the adversary to subvert everyone, all the time.

8.2 Message Indistinguishability for Mixnets

Recipients of messages might want to reply to a sender while preserving anonymity for both. How can a recipient of an anonymous message reply, without uncovering the real address of the sender? Through using a cryptographic reply address.
Alice messages:

$$\mathcal{A} \xrightarrow{M_1,\{M_2,k_1,\{A,k_2,\{K\}_A\}_{M_2}\}_{M_1}} \mathcal{B}$$

Where $k_1 = H(K,1)$, $k_2 = H(K,2)$
Bob replies:

$$B \xrightarrow{\{M_2,k_1,\{A,k_2,\{K\}_A\}_{M_2}\}_{M_1},\text{Msg}} M_1$$

$$M_1 \xrightarrow{\{A,k_2,\{K\}_A\}_{M_2},\{\text{Msg}\}_{k_1}} M_2$$

$$M_2 \xrightarrow{\{K\}_A,\{\{\text{Msg}\}_{k_1}\}_{k_2}} A$$

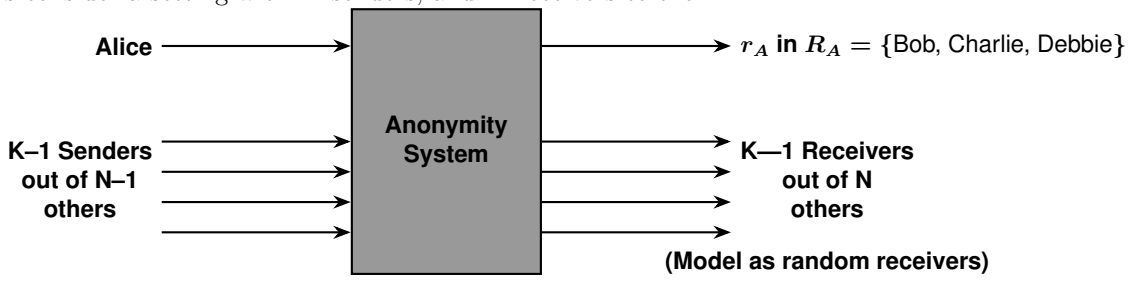
9 Basic Traffic Analysis

Traffic analysis allows information to be inferred from patterns of encrypted data. Assume that content is encrypted, traffic is anonymized, we want to conduct traffic analysis of "hardened targets". Even perfect anonymity systems leak information when the participants change.

Consider the following setting;

- Let N senders/receivers where Alice is one of them.
- Alice messages a small number of friends:
  - R_A in $\{\text{Bob}, \text{Charlie}, \text{Debbie}\}$
  - Through a mix/DC-network
  - Assume perfect anonymity of size K.
- Can we infer Alice's friends?

Let's consider a setting with K senders, and K receivers to the mix.



Alice sends a single message to one of her friends. The anonymity set has size K, and an entropy metric $E_A = \log k$.

We can assume that communication continues over many rounds. Therefore the rounds which Alice participates in will output a message to her friends, which an adversary may take and be able to infer her set of friends.

10 Low Latency Anonymity Systems

Tor and many similar systems utilise onion routing, allowing for anonymity for web browsing and other low-latency applications.

- Tor anonymises streams of messages, and by default, Tor network traffic travels over three hops (entry, middle, exit)
- Mixnets let users choose short paths and relays a bi-directional stream of traffic to Bob.
- For tor onion services there are six hops to provide anonymity to both participants.

Tor is used by a wide range of people, which is essential for it to maintain security.

- **Ordinary people**: to avoid unscrupulous marketers, protect children, research sensitive topics
- **Militaries and law enforcement**: to protect field agents and sources, intelligence gathering, decentralise services
- **Journalists**: preserve safety of journalists working in hostile regimes, resist internet censorship

The layered encryption (telescoping onion routing) that Tor uses is for confidentiality and integrity. Layered encryption prevents linking of input and output flows based on traffic content.

Network information is collected by directory authorities and distributed through mirrors.

- Tor nodes **publish** their information (descriptors) to the directory **authorities**.
- These directory authorities negotiate a **consensus** listing all available Tor nodes, and **sign** under keys of **all participating** directory authorities.
- Directory **mirrors** (most of the Tor nodes) **connect** to directory authorities to retrieve consensus.
- Tor clients **connect** to mirrors to receive **up-to-date** consensus, or bootstrap by connecting to directory authority directly.
- Tor clients **verify** digital signatures on consensus.

The Tor network has around 7,000 relays used for normal traffic, and 2,000 bridges (for censorship resistance) (Fig. 1). This number of relays has stayed more or less constant, and follows from economics: it is cheaper to get more bandwidth than more servers.

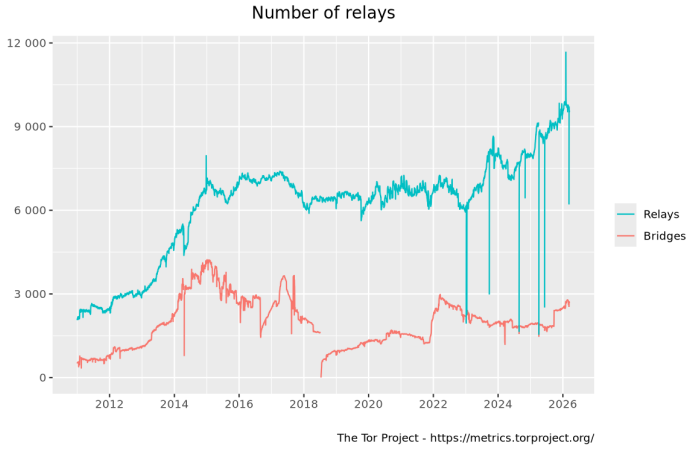


Figure 1: Number of Tor relays over time.

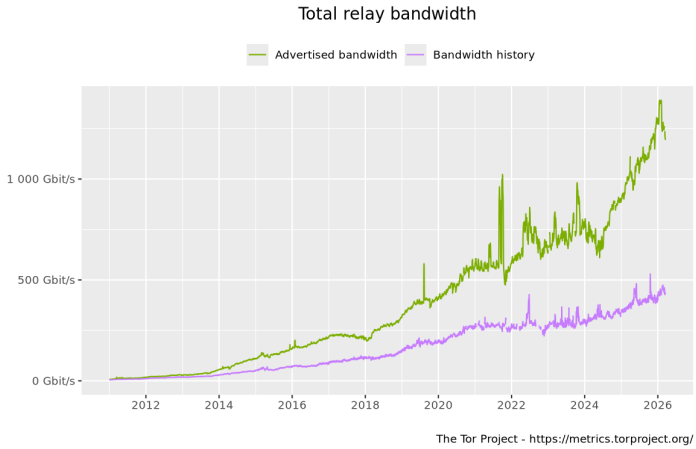


Figure 2: Tor relay bandwidth over time.

Tor has around 600 Gbit/s capacity and about 300 Gbit/s usage (Fig. 2). Relay bandwidth is affected by widescale Internet disruption. Large additions to the network are suspicious and are sometimes blocked if not expected.

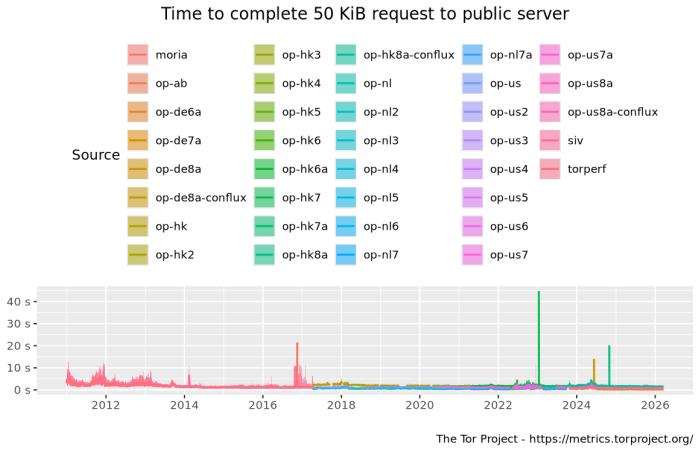


Figure 3: Download time for 50KiB request over time.

From Fig. 3,we see download performance has significantly improved due to better load balancing. We can see that load balancing used to be based on unreliable data, but the algorithms and implementation have since improved. Note that the **variance** in download times is just as important as raw bandwidth.

11 Attacking Low-Latency Anonymity

Onion routing and mix networks have different vulnerability profiles. We only need to setup the route once per connection. This is done for many cells and saves on public key operations. Low latency networks are more vulnerable to passive attacks.

A stream tracing attack uses timing patterns to follow a stream through the network. Given an adversary observing all inputs and outputs of an onion router, he is able to link the incoming and outgoing connections (to trace a message from Alice to Bob). Because the timing of packets are correlated, this (**correlation**) allows matching of an input stream to output stream based on packet counts. On the other hand, **template matching** takes into account the probability distribution of packet delays. using the input and delay curve to make a template to predict what the output will be.

Tor offers less security than high-latency systems but attracts more users, therefore tracing attacks become too expensive to foil, as such, Tor would not be able to withstand a global passive adversary. Partial adversaries can potentially view some of the network, and likely control some of the nodes. But as long as the adversary cannot see the first and last node of the connection, i.e, if c is a fraction of corrupt servers, the compromise probability is $O(c^2)$. Therefore, there is little point in making routes very long.

Anonymous communication introduces particular requirements for cryptography. For example, the encryption of input and output streams under different keys must provide bitwise unlinkability (as in mixnets). Authenticated DH encryption allows one-sided authentication; i.e, Alice remains anonymous. Though Alice still needs to know the signature keys of the onion routers, which means scalability issues, but guarantees forward secrecy.

In general, anonymous communications are more challenging than E2EE.

- Anonymity requires a crowd
- Mixnets offer practical anonymous messaging by offering
  - Bitwise unlinkability and traffic analysis resistance
  - Use of cryptography, distributed networks, and robust partial checking.
- Onion routing - supports interactive streams, and withstands a partial adversary.

Privacy-friendly computation

Privacy Enhancing Technologies (COMP0061)
Steven Murdoch | Sem 2 25/26 | 26-30 Jan

---

12 Privacy-friendly computation

Hiding information from third parties is a (relatively) easy problem. But it is harder to hide information from your communication partners.

12.1 Yao's Millionaire Problem

Alice and Bob do not trust each other with their secrets but still want to jointly compute on them. They may not even trust each other to perform any computations correctly, as such, their observations from the protocol should not disclose any information on the inputs.

consider f with n inputs x_i from distinct parties returning $y = f(x_1, \ldots, x_n)$. We want to compute y but not learn anything more about x_i than what we would learn by learning y, despite observations o from the protocol. e.g,

$$\Pr[x_i|o, y, x_j] = \Pr[x_i|y, x_j]$$

Strawman: use a trusted third party (TTP).

Why not use a TTP? Even if such a party existed, the business **costs** may not be feasible. You cannot trust that the business is not **corrupted**. There can exist legal **compulsions** to reveal your secret. TTPs can get **compromised**.
In theory, any function can be computed without a TTP.

12.2 Idea: implement NAND privately, to compute anything

If we can express binary digits, compute addition and multiplication privately, we can compute all circuits.

Homomorphic encryption (HE) and secret sharing (SS) are two approaches to private computation. Where HE allows for operations on ciphertexts to produce a resulting ciphertext that can be decrypted as the same homomorphic computation on the equivalent plaintexts. SS expresses $0, 1$ as shares distributed between users where addition and multiplication are performed as protocols on shares.

13 Homomorphic encryption

Additively homomorphic public-key encryption allows addition of encrypted data.

The goal is defining functions for:

- `GenKey`
- `Encrypt`
- `Decrypt`
- `Add`
- `Multiply` (not in this example)

We are performing operations on a cyclic group, just as with public key encryption. So we define two elements g, h that are generators of a cyclic group within which the DLP is believed to be hard.

Generators mean g^i may lead to all group elements. The DLP means that given $g, x \to g^x$ is easy to compute, but for $g, g^x \to x$ is hard.

13.1 Modifying ElGamal

ElGamal can be modified to add encrypted values. **The Scheme:**

- Public: g, h (and group params)
- Key generation: generate a random x where $0 < x <$ order of group.
- sk $= x$, pk $= g^x$.

- Encryption of m with pk: $E(m;k) = (g^k, g^{xk}h^m)$ where k is random.
- Decryption of (a,b) with $x : m = \log_h(b(a^x)^{-1})(= \log_h g^{xk}h^m/g^xk)$

Using this encryption we can add ciphertexts to get the encryption of the sum. The homomorphic properties include:

- **Addition**: $E(m_0;k_0) = (a_0,b_0)$ and $E(m_1;k_1) = (a_1,b_1)$

  $$E(m_0+m_1;k_0+k_1) = (a_0a_1, b_0b_1) = (g^{k_0}g^{k_1}, g^{xk_0}h^{m_0}g^{xk_1}h^{m_1}) = (g^{k_0+k_1}, g^{x(k_0+k_1)}h^{(m_0+m_1)})$$

- **multiplication**: $E(m_0;k_0) = (a_0,b_0)$ with constant c:

  $$E(cm_0;ck_0) = ((a_0)^c, (b_0)^c)$$

Private elections/polls can be implemented through such a system. e.g, A poll asks a number of participants whether they prefer "red" or "blue". How many said "red" and how many "blue"?

Each participant submits a ciphertext for both "red" and "blue" to an authority, who homomorphically adds them.

13.1.1 Limitations

The domain of plaintexts is small (up to number of participants), so decryption by enumeration is cheap. But who's public key is used? and who owns the decryption key? Afterall a single decryptor is essentially a TTP, and if no-one can decrypt the scheme is usseless.

13.2 Threshold Decryption

Threshold decryption avoids having a central party which can decrypt the ciphertext. It is generally better that no one has the secret key as no TTP is needed.

The secret key is distributed across many different people, where each have to contribute to the decryption. This means that even if it is below the threshold, the remaining participants cannot decrypt.

- **Private keys**: $x_1, \ldots, x_n$
- **Public key**: $g^{x_1+\ldots+x_n}$
- **Decryption** (a,b): $m = b/a^{x_1}/a^{x_2}/\ldots/a^{x_n}$

14 Beyond ElGamal

There are alternatives to ElGamal that have different limitations.

- **RSA**: multiplicative but no addition.
- **Paillier**: additive homomorphism only and based on RSA (large ciphertexts so is slow)
- **Pairings on EC**: addition and 1 multiplication.

More advanced schemes can do a lot more, but come at significant performance cost. Craig Gentry first introduced a fully homomorphic scheme in 2009, but was inefficient for realistic applications.

The simplicity in architecture of HE allows for participants to provide encrypted inputs. Where any party may compute the function. Secret functions can exist as part of the function itself which remain secret, and the service can perform whatever operations without telling any party.

Linear operations are fairly efficient and multiplication *can* be relatively fast along with any function with shallow depth. However, limitations exist with expressing computations as boolean circuits makes them much more expensive. Efficiency is also a problem as every bit $\rightarrow$ 160bit, 1024bit, ..., 30Kbs. Further there is the problem of decryption integrity.

14.1 Attack on HE

If the party doing the computation is actively malicious, then they can also decrypt the messages. A malicious party can simply ask the threshold decryption parties to decrypt a secret, not the output of the computation. (decryption oracle attack) This means we cannot achieve confidentiality without integrity.

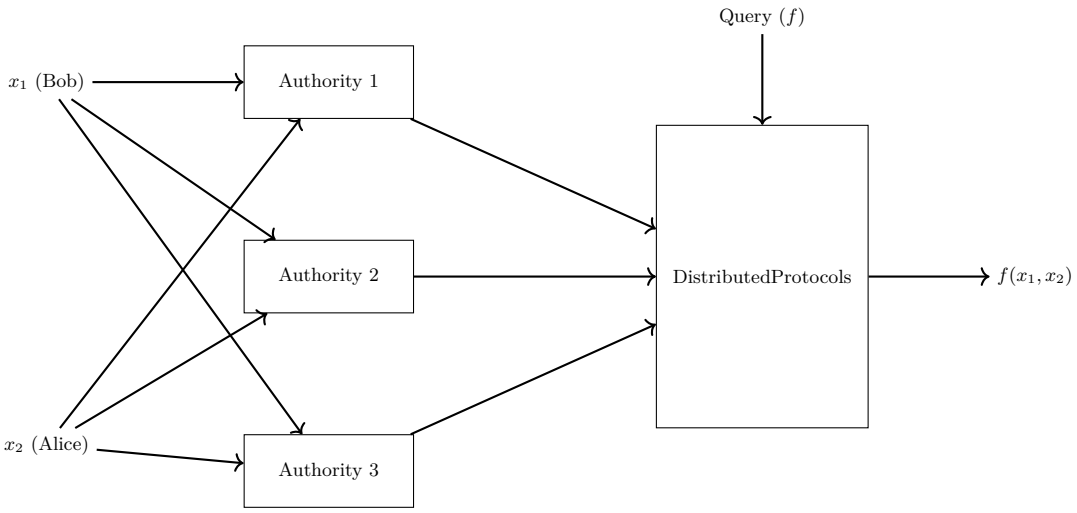
Solutions include:

- The central party needs to prove that the output of the computation was indeed correct.
- Public computation so that anyone can verify. However, this is computationally expensive.
- Or private computation, but has not yet been tried, not until the private information can be turned into data.

15 Secret Sharing

Secrets can be shared across authorities and used to compute functions. The idea is to share the secret amongst multiple authorities, who use protocols to transform these shares into secrets.

15.1 Secret Sharing Architecture



The basic scheme is to split the secret into a number of shares. A share of secret x is denoted as "$<x>$", if we add all shares $<x>$ mod p we get x. e.g;

- Prime $p = 2, x = 1$
- Shares $<x>$ are $\{1, 1, 0, 1, 0\}$
- We check $1 + 1 + 0 + 1 + 0 \mod 2 = 1$

Additive secret sharing like this is simple and based on a natural additive homomorphism. Adding $<a>$ and $<b>$, each authority simply adds like: $<c> = <a + b> = <a> + <b> \mod p$

We can also add or multiply by a public constant value k. First, we split k into $<k>$ as $\{0, 0, \ldots, 0, k\}$ and do addition between $<k>$ and $<a>$. Then, each authority may privately compute the multiplication $<a>$ by k as; $<c> = <ka> = k<a>$.

Multiplication of secrets is more complex than multiplication by a public value as it requires pre-computed values and the protocol interacts with authorities. The pre-computed value is an issue as it requires an additional component independent from the function f and should not simply be derived by a TTP.

we can have precomputed $<a>$, $<b>$, $<c>$ such that $<c>=<ab>$ To multiply $<x>$ and $<y>$;

- We get a fresh $<a>$, $<b>$, $<c>$.
- Compute $<e>=<x>+<a>$
- Compute $<d>=<y>+<b>$
- we publish $<e>$ and $<d>$ to get e and d.
- Compute $<z> = <xy> = <c> - e<b> - d<a> + ed$ (which is linear).

Logic gates can also be implemented through SS mechanisms, however, it is not necessarily more efficient. It is much faster to do linear operations on shares of the actual secrets rather than circuit bits.

16 Integrity in Secret Sharing

The traditional approach is to perform a zero-knowledge proof that what it publishes is correct, but this is expensive.

The solution is we can use Message Authentication Codes (MACS), such that each secret share is associated with a MAC. We maintain the MAC through computations, but never reveal the MAC.

One MPC protocol is SPDZ, who's MACs share a secret v as $<v>$, and each share $<a>$ has a MAC share $<va>$. This way authorities can know $<v>$ but no one can know and derive v.

MACs are carried around with operations to ensure integrity. **Addition** works simply by adding secrets and MACs, while **Multiplication** requires pre-shared values to also have a MAC, otherwise it is the same technique. Constants are included and easy to endorse (for k compute $<k>$, $<vk>=k<v>$).

But how can we check the MAC without revealing it? The intuition is that since every value has a MAC associated with it, and the operations preserve the MAC, everything that is declassified needs to have its MAC checked. This is the point where authorities interact, as a malicious authority may give a wrong share. We can check that the relation holds for a fixed MAC key $<v>$, by checking that a, $<va>$. And, the relation $a<v>=<va>$, without ever revealing $<v>$

We can also check the MAC in parallel:

- as $k<v>=<kv>$
- which is the same as $k<v>-<kv>=<0>$.
- also the same as $w(k<v> - <kv>) = <0>$.

Therefore $\text{sum}_i w_i(k_i<v> - <k_i v>) = <0>$ can be done many times in parallel.

The integrity protocol can run in parallel with the computation. First, we commit to the intermediate and final results (but do not reveal it). Then, jointly generate random w_i. For all intermediate results, we compute:

- $<c> = \text{sum}_i w_i(k_i<v> - <k_i v>) = <0>$
- Reveal $<c>$ and check it is $<0>$ guaranteeing no information leakage.

We may then reveal the final results $<c> = \text{sum}_i w_i(k_i<v> - <k_i v>) = <0>$ with the same process guaranteeing the actual results are also correct.

16.1 Secret Sharing: Cost of Integrity

We see here that integrity comes at a relatively low cost.

- Two shares instead of one.
- Triplets with MACs for multiplication.
- Two checks per computation (that are batched).

SS uses cheap operations but requires lots of network interactions. There can be a vast number of triplets needed (one per gate), and there may arise complications about generating them. Further, circuits still are inefficiently expressed, and the computations themselves cannot be secret.

17 Summary

Overall private computations are feasible in many situations but do add an overhead.

Private computations are generally more expensive than regularly computing something. Computing linear operations are cheap but non-linear will cost more, limitinge the depth for non-linear operations can help with efficiency. Integrity is generally a problem for confidentiality so using MACs is useful.

Zero-Knowledge Proofs

Privacy Enhancing Technologies (COMP0061)

Steven Murdoch | Sem 2 25/26 | 2-6 Feb

---

18 From Confidentiality to Integrity and Authentication

E2E encryption (Section 3) allows us to protect messaging content, while **anonymous communications** (Section 7) also allows us to protect the meta-data such that association between communicating parties is not revealed.

Confidentiality for computation can be achieved through privacy-friendly computation such as **HE** (Section 13) and **SS** (Section 15) to protect private inputs and intermediate results of computations on private data.

This lecture focuses on protecting Integrity/Authenticity and Privacy through **zero-knowledge** (ZK) proofs which protect integrity without revealing any information on the data. This lecture also covers **credentials** which protect authenticity without revealing private data.

18.1 ZK overview

A zero-knowledge proof shows a statement on secrets without revealing the secret.

It is a protocol between a *prover* and a *verifier*, which allows the prover to convince the verifier of a statement on secrets (allowing for integrity and soundness), without revealing anything about the secret (maintaining privacy and zero-knowledge principle).

This property intuitively feels impossible but offer practical real world applications. These properties may rely on the verifier being computationally bound but some proofs have been proposed to offer unconditional privacy.

Proving knowledge of a signature key is an example of a zero-knowledge proof. e.g, The signer needs to convince the verifier that they know the signature key (sk), and use it to sign a message, without revealing this secret key. All have the verification key (vk) and are convinced of this statement.

In practice,

- $\text{sk} = x \in \mathbb{Z}_q$
- $\text{vk} = g^x$
- where g is a public fixed generator of group $\mathbb{G}$ in which the DH problem is hard.
- Do you know the x such that $\text{vk} = g^x$ and public message?

Homomorphic encryption can require applying a ZK proof, assume g, h public generators of $\mathbb{G}$ where the DH problem is hard.

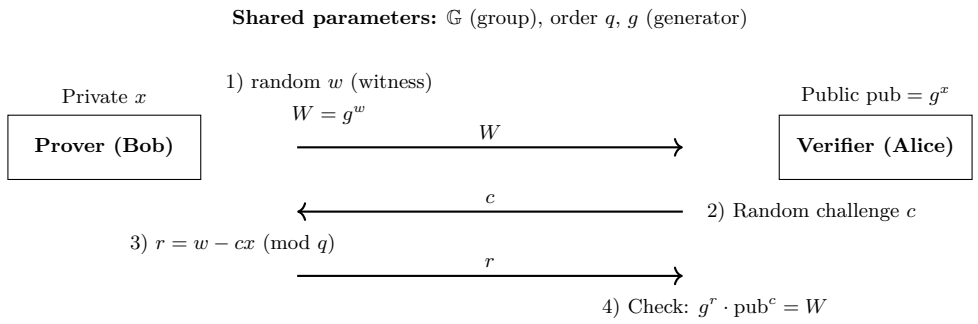
- $\text{GenKey}() : \text{priv} = x \xleftarrow{\$} \mathbb{Z}_q, \text{pub} = g^x, \text{output}(\text{priv}, \text{pub})$
- $\text{Encrypt}(m, \text{pub}) : \text{output}(g^k, \text{pub}^k h^m)$ for $k \xleftarrow{\$} \mathbb{Z}_q$.
- $\text{Encrypt}(m, \text{pub}) : \text{output}(b \cdot a^{-\text{priv}})$

But how do we ensure that m is a valid vote (e.g, $m \in \{0, 1\}$)? And how do we secure threshold decryption? Threshold decryption in an honest-but-curious model has each authority run $\text{GenKey}() : (\text{priv}_i, \text{pub}_i)$ and publishes pub_i. Then the global public key is $\text{pub}_{\text{global}} = \prod_i \text{pub}_i = g^{\Sigma_i x_i}$.

But if the last authority n is malicious, they could simply derive their keys $\text{priv}_n, \text{pub}_n = \text{GenKey}()$, and give others $\text{pub}'_n = \frac{\text{pub}_n}{\prod_{i-1} \text{pub}_i}$. Resulting in the global public key being g^{priv_n} allowing n to decrypt all ciphertexts on its own.

The solution is we require all authorities to publish pub_i and a ZKP that they know a private priv such that $\text{pub} = g^{\text{priv}}$.

18.2 The Schnorr ID Protocol



18.3 3 key properties for ZK

- **Completeness:** Does the verification work?
- **Integrity/Soundness:** Does it prove Bob knows x?
- **Privacy/Zero-Knowledge:** Are we leaking any information about x?

19 Implementing ZKPs

19.1 Completeness

We first verify that the Schnorr identification protocol satisfies the verification equation.

The verifier checks: $g^r \cdot \text{pub}^c = W$

Recall these definitions:

$$\begin{aligned} r &= w - cx \pmod q, \\ \text{pub} &= g^x, \\ W &= g^w. \end{aligned}$$

Verification:

$$\begin{aligned} g^r \cdot \text{pub}^c &= g^{w-cx} \cdot (g^x)^c \\ &= g^{w-cx} \cdot g^{cx} \\ &= g^w \\ &= W. \end{aligned}$$

19.2 Integrity/Soundness

We now argue that a prover who convinces the verifier with non-negligible probability must know the secret x.

The argument proceeds via a *rewinding* technique. Suppose the verifier can issue two different challenges c and c' for the same commitment $W = g^w$. A successful prover must then produce:

$$\begin{aligned} r &= w - cx, \\ r' &= w - c'x. \end{aligned}$$

Subtracting the equations eliminates w:

$$r - r' = (c' - c)x.$$

Solving for x:

$$x = \frac{r - r'}{c' - c} \pmod q.$$

Therefore, if a prover can answer correctly for two distinct challenges with the same commitment, one can efficiently extract x. Hence, any prover that succeeds with probability greater than $1/2$ must "know" the secret.

19.3 Privacy and Zero-Knowledge

Finally, we have to show that the protocol does not leak information about x.

The idea is that transcripts of the protocol can be simulated without knowledge of x. A transcript consists of (W, c, r) satisfying:

$$W = g^r \cdot \text{pub}^c.$$

A simulator can generate such a transcript as follows:

- Choose $r'' \xleftarrow{\$} \mathbb{Z}_q$ uniformly at random,
- Choose $c'' \xleftarrow{\$} \mathbb{Z}_q$ uniformly at random,
- Compute

$$W'' = g^{r''} \cdot \text{pub}^{c''}.$$

The triplet (W'', c'', r'') satisfies the verification equation by construction:

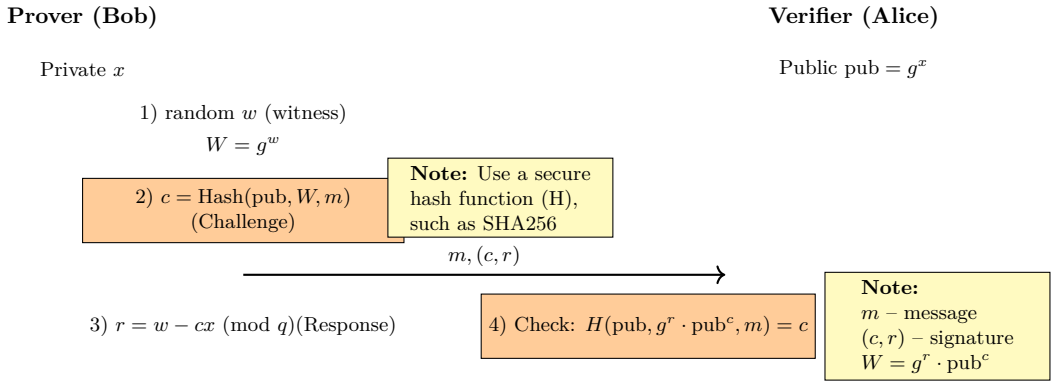
$$g^{r''} \cdot \text{pub}^{c''} = W''.$$

Moreover, this simulated transcript is indistinguishable from a real execution of the protocol. Therefore, the protocol leaks no information about x beyond the validity of the statement.

Remark. The prover must commit to W before seeing the challenge c, which prevents cheating and is essential for both soundness and zero-knowledge.

19.4 Non-Interactive Digital Signature

Digital Schnorr Signatures can be turned non-interactive through the Fiat-Shamir heuristic.



Fiat-Shamir is a heuristic based on the security of the hash function (e.g, SHA256). specifically, pre-image resistance meaning that given $Y = H(\ldots y \ldots)$, it is difficult to find a y.

Remember that to build a fake proof, a simulator determines r'' and c'', then W''. But because $c = H(\ldots, W, \ldots)$, the prover cannot first set c'' and then W''. Here, the use of the hash function forces causality (i.e, prover must first set W then c).

Schnorr enables Bob to prove knowledge of x. Since Alice knows pub, Bob may interactively prove he knows the private key x `s.t.` $\text{pub} = g^x$.

- Verification equation ensures **completeness** through extraction of the secret under rewinding.
- The potential existence of a simulator provides **privacy**.
- Causality ensures the proof's **soundness**.

20 Extending ZKPs

Now we know how to build an efficient signature scheme based on the DL problem, and we also can prove knowledge of the key $\text{pub}_i = g^{\text{priv}_i}$ using Schnorr's (or NIZK).

ZK can be more than just knowledge, we can also encode linear relationships e.g,

$$z = 3x + 4y - 10$$

or

$$z = xy$$

We could prove statements about any encrypted/committed secrets. We could also prove all the inputs and the intermediate values of a network of NAND gates. Therefore we can prove "any logic circuit". In practice, there are more efficient proofs for interesting statements.

20.1 Equality proof for DL

g and h are generators of $\mathbb{G}$ or order q (in which DL is hard). Consider $P_1 = g^x$ and $P_2 = h^x$. Bob wants to prove the statement:

$$\text{NIZK}\{(x) : P_1 = g^x \text{ and } P_2 = h^x\}$$

The protocol involves composing two Schnorr signatures with the same witness/response. Where Bob computes:

- $W_1 = g^w, W_2 = h^w$ where $w \xleftarrow{\$} \mathbb{Z}_q$
- $c = H(g, h, P_1, P_2, W_1, W_2)$
- $r = w - cx$
- send (c, r) to Alice (verifier)

Alice can then verify $H(h, g, P_1, P_2, g^r \cdot P_1^c, h^r \cdot P_2^c) == c$

Encryption protects secrets whereas commitments protects changing history, this allows Bob to encrypt a secret under the public key of Alice. ZKP may prove some statements about the encrypted values (whether $x \in \{0, 1\}$, etc.).

Commitments are like a safe that can only be opened by the person with the key.

They have two operations:

- $C, o = \text{Commit}(v)$
- $\text{Reveal}(C, v, o)$

and two properties:

- **Hiding:** a commitment reveals nothing about the committed value
- **Binding:** one can only open the commitment and reveal the committed value

20.2 Chaum-Pedersen Commitments

These allow ZK proofs of statements about committed values. They include the following shared parameters:

- Group $\mathbb{G}$ with order q (assuming DL is hard)
- Generators g and h of $\mathbb{G}$.

Defined terms:

- v - value to be committed
- C - the commitment
- o - a random nonce $\in \mathbb{Z}_q$

$\text{Commit}(v)$: outputs $C = g^v h^o$, and $\text{Reveal}(C, v, o)$: returns $g^v h^o == C$.
Bob (prover) produces:

- $w_1, w_2 \xleftarrow{\$} \mathbb{Z}_q; W = g^{w_1} h^{w_2}$
- $c = H(g, h, C, W)$
- $r_1 = w_1 - cv$
- $r_2 = w_2 - co$
- Send (c, r_1, r_2)

Alice (verifier) checks:

- $H(g, h, C, g^{r_1} h^{r_2} C^c) == c$ (because $g^{w_1 - cv} h^{w_2 - co} C^c \Rightarrow (g^{w_1} h^{w_2})(g^v h^o)^{-c} C^c \Rightarrow W C^{-c} C^c \Rightarrow W$)

20.3 Proving DL representations of C in ZK

We can also show that Bob knows a DL representation of $C = g_1^{x_1} g_2^{x_2} g_3^{x_3} \ldots$. Assuming that $g_1, \ldots, g_i$ generators of $\mathbb{G}$, and $x1, \ldots, x_i$ secrets. Commitment $C = \prod_i g_i^{x_i}$.
Bob (prover) produces:

- $\ldots w_i \ldots \xleftarrow{\$} \mathbb{Z}_q; W = \prod_i g_i^{w_i}$
- $c = H(\ldots g_i \ldots, C, W)$
- $r_i = w_i - cx_i$
- Send $(c, \ldots r_i \ldots)$

Alice (verifier) checks:

- $H(\ldots g_i \ldots, C, (\prod_i g_i^{x_i})C^c) == c$

We can prove the statement:

$$\text{NIZK}\{(x_i) : C = \prod_i g_i^{x_i}\}$$

This effectively means that if you can reduce a ZK proof to a DL representation proof, you can prove it. Remember that equality can be proven by simply using the same w, r for equal values.

20.4 Proving linear relationships between secrets in ZK

consider 3 commitments on secrets $x; y; z = a \cdot x + b \cdot y + d \pmod q$ where;

- $C_x = g^x h^{o_1}$
- $C_y = g^y h^{o_2}$
- $C_z = g^z h^{o_3}$

To prove $\text{NIZK}\{(x, y, z, o_i) : C_x = g^x h^{o_1}, C_y = g^y h^{o_2}, C_z = g^z h^{o_3}, \text{ and } z = a \cdot x + b \cdot y + d\}$, we need to substitute the linear relationship for z.

$$C_z = g^z h^{o_3} = g^{a \cdot x + b \cdot y + d} h^{o_3} \equiv (c_z g^{-d}) = (g^a)^x (g^b)^y h^{o_3}$$

thus it is sufficient to prove:

$$\text{NIZK}\{(x, y, z, o_i) : C_x = g^x h^{o_1}, C_y = g^y h^{o_2}, C_z = g^z h^{o_3}, \text{ and } (C_z g^{-d}) = (g^a)^x (g^b)^y h^{o_3}\}$$

using DL representations and equality proofs.

20.4.1 Some examples of NIZK proofs

Example 1:
Let $\text{NIZK}\{(x, y, z, o) : C = g_1^x g_2^y g_3^z g_4^o \text{ and } z = 10x - 5y - 1\}$, to prove the statement, we can first substitute z into C

$$\begin{aligned} C &= g_1^x g_2^y g_3^{10x-5y-1} g_4^o \\ C &= (g_1 g_3^{10})^x (g_2 g_3^{-5})^y g_3^{-1} g_4^o \\ Cg_3 &= (g_1 g_3^{10})^x (g_2 g_3^{-5})^y g_4^o \end{aligned}$$

Let

$$C' = Cg_3, G_x = g_1 g_3^{10}, G_y = g_2 g_3^{-5} G_o = g_4$$

Then the original statement becomes:

$$\text{NIZK}\{(x, y, o) : C' = G_x^x G_y^y G_o^o\}$$

Witness z no longer needs to be proven separately, so the prover generates their $r_x, r_y, r_o \xleftarrow{\$} \mathbb{Z}$ and constructs $A = G_x^x G_y^y G_o^o$.
The prover then computes the Fiat-Shamir hash $c = H(G, g_1, g_2, g_3, g_4, C, C', G_x, G_y, G_o, A)$.
Then computes the responses

$$\begin{aligned} s_x &= r_x + cx \pmod q \\ s_y &= r_y + cy \pmod q \\ s_o &= r_0 + co \pmod q \end{aligned}$$

The proof is

$$\pi = (A, s_x, s_y, s_o).$$

The **verifier** can then compute

$$C' = Cg_3, G_x = g_1 g_3^{10}, G_y = g_2 g_3^{-5}, G_o = g_4$$

Then constructs $c = H(\ldots)$, and accepts if $G_x^{s_x} G_y^{s_y} G_o^{s_o} = A(C')^c$. Because,

$$G_x^{s_x} G_y^{s_y} G_o^{s_o} = G_x^{r_x+cx} G_y^{r_y+cy} G_o^{r_o+co} = A(G_x^x G_y^y G_o^o) = A(C')^c$$

Therefore the verifier is convinced that the prover knows such x, y, o such that $Cg_3 = (g_1 g_3^{10})^x (g_2 g_3^{-5})^y g_4^o$, which is equivalent to knowing $z = 10x - 5y - 1$.

Example 2:
Let NIZK$\{(x, y, z, o) : C = g_1^x g_2^y g_3^z g_4^o$ and $x = y$ and $z = 10\}$. First, substitute $y = x$ and $z = 10$ into C.

$$C = g_1^x g_2^x g_3^{10} g_4^o C \qquad = (g_1 g_2)^x g_3^{10} g_4^o C g_3^{-10} = (g_1 g_2)^x g_4^o$$

Then

$$G_x = g_1 g_2, g_o = g_4, C'' = C g_3^{-10}$$

Therefore the statement becomes

$$\text{NIZK}\{(x, o) : C'' = G_x^x G_o^o\}$$

The prover knows x, o and sample $r_x, r_o \xleftarrow{\$} \mathbb{Z}$, then computes $A = G_x^{r_x} G_o^{r_o}$. Then computes the challenge

$$c = H(G, g_1, g_2, g_3, g_4, C, C'', G_x, G_o, A)$$

Computes the responses

$$s_x = r_x + cx \pmod q s_o \qquad = r_o + co \pmod q$$

The proof is

$$\pi = (A, s_x, s_o).$$

The **verifier** then computes $C'' = Cg_3^{-10}, G_x = g_1 g_2, G_o = g_4$, and recomputes $c = H(\ldots)$.
The verifier only accepts if

$$G_x^{s_x} G_o^{s_o} = A(C'')^c$$

Equivalently,

$$(g_1 g_2)^{s_x} g_4^{s_0} = A(C g_3^{-10})^c$$

Therefore verifies that the prover has knowledge of x, o `s.t.` $Cg_c^{-10} = (g_1 g_2)^x g_4^o$. Which is equivalent to $C = g_1^x g_2^x g_3^{10} g_4^o$, knowing x, y, z which satisfies $x = y, z = 10$.

Multi-Party Computation and Private Set Intersection

Privacy Enhancing Technologies (COMP0061)
Steven Murdoch | Sem 2 25/26 | 9-13 Feb

---

21 Secure Multi-party Computation (MPC)

Suppose we have multiple parties that have some private data, and do not trust the rest of the parties, but want to compute some function f with their private data. How can this be done to not reveal each parties input to each other, while protecting the privacy of each input?

MPC allows parties to run a function on their private inputs. In an MPC, privacy must be preserved even if some of the parties are not good actors (or are adversaries).

21.1 MPC Adversaries

In general there are two types of adversaries:

- Passive (semi-honest): will follow the protocol's instruction, but tries to learn extra information (or parties' inputs).
- Active (malicious): acts arbitrarily (and may not follow the protocol's instruction). May lie about its input or even quit at any point. They may try to learn more information than an honest party would do and affect the computation's results.

21.2 MPC Security

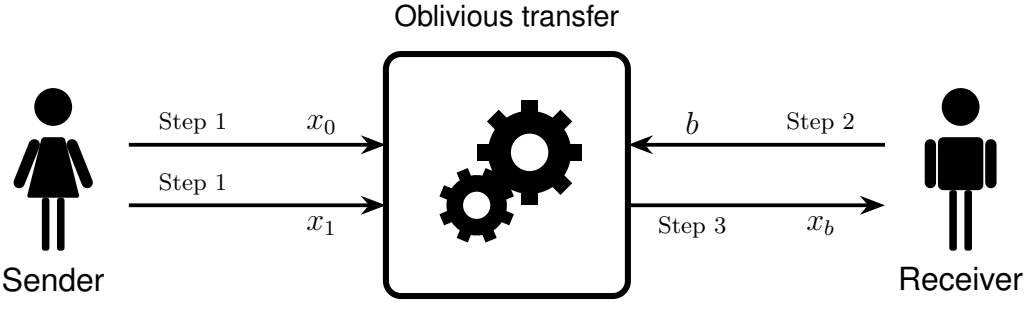
We may define an MPC's security against their modelled adversary. MPCs safe against only passive adversaries may be more efficient, and have a simpler design such that it is easier to analyse.

MPCs modelled against active adversaries are generally less efficient, cost more to compute, are harder to analyse because of their complex design, but offer greater security against a stronger adversary.

In modern cryptography, the efficiency of protocols play is a vital determining factor. Designers of MPC protocols try to make them secure and efficient, An MPC design can be very complex the protocol has to be highly efficient.

21.3 Oblivious Transfer (OT)

A cryptographic protocol which involves two parties (sender and receiver). A sender transfers one of the potentially many pieces of information to a receiver but remains *oblivious* as to what piece (if any) has been transferred. The receiver learns niothing about the senders' other pieces.



b is a binary value $\in \{0, 1\}$, the sender learns nothing about b and the receiver learns nothing about the. other input of the sender, i.e, x_{1-b}.

In general, oblivious transfers are based on public-key primitives; these primitives are invoked proportionate with the number of inputs, which could impose a high cost. Some efficient oblivious transfers require a small number of public-key operations and the rest of the operations are based on symmetric-key primitives, which ultimately lead to lower costs.

21.4 Rigorous approach to Security

Generally we want to define the adversary, are they active or passive? What is their computational ability? Are there potentially many adversaries colluding in the protocol?

We need a defined network model, i.e, do the parties use a secure channel to communicate? Would the adversary change the order of messages exchanged?

For most encryption or signature schemes, we use game-based definitions, a la *Introduction to Cryptography*. For MPC protocols, simulation-based definitions can be used.

21.5 Simulation-based definition

Based on real and ideal models (world models), an ideal model has:

- The scheme's participants (or parties) send their inputs to a trusted party, who computes the function and sends the outputs.
- The adversary is among the parties and (indirectly) deals with the trusted party.

real models have:

- The parties run a real (MPC) protocol with no trusted party's help.
- The adversary is among the parties and deals with the protocol and other parties.

Generally, a protocol is secure if any attack on a real model can be performed (simulated) in the ideal model. Because no attacks can be carried out in the ideal model, security is implied.

The behaviour of an MPC protocol executing in the real model is compared against the behaviour of a simulator that interacts with an ideal process. The role of the simulator is to simulate the view of corrupted parties, given their inputs and outputs. Intuitively, if the behaviour of these two models are indistinguishable, then the protocol is at least as secure as the ideal process, or it must hold that the output distribution of the honest parties and the simulator in an ideal model is computationally indistinguishable from the output of the honest parties and adversary in a real model.

The phases of designing a simulation-based definition of security of an MPC are as follows:

1. Construct the ideal model
2. Design the protocol (real model)
3. Construct a simulator (in the ideal model)
4. Prove security by showing indistinguishability between the two models.

22 Common MPC Tools and Techniques

- Homomorphic Encryption
- Commitment Schemes
- Secret Sharing
- Trusted Execution Environment
- Blockchain Technologies

22.1 Homomorphic Encryption (brief recap)

HE allows the execution of computations on encrypted data without decrypting it.

FHE supports arbitrary computations on ciphertexts but imposes significant computation and communication costs.

SHE supports limited computations on ciphertexts but is more efficient than FHE.

We might want to use HE in MPC because it enables parties to run a computation on their encrypted data without learning anything about the parties input. After the computation result is decrypted, the parties would learn only the result. Thus, the inputs' privacy would be preserved.

22.2 Commitment Schemes

Commitment is a vital primitive in many cryptographic protocols (including MPC). It allows a sender to *commit* to a secret value, and reveal it some time later to a receiver.

Commitment schemes have two phases (or algorithms):

1. **Commit Phase**: Sender commits to a message m as $\texttt{com}(m, d) = h$ involving a secret value d. The commitment h is sent to the receiver.

2. **Open Phase**: sender sends the opening $\ddot{p} = (m, d)$ to the receiver who verifies its correctness $\texttt{ver}(h, \ddot{p}) \stackrel{?}{=} 1$ and accepts if the output is 1.

Commitment schemes must satisfy:

- **hiding**: No adversary can learn any information about the commitment m, until h is opened.
- **binding**: No adversary can open a commitment h to different values: $\ddot{p}' = (m', d')$ than that used in the commit phase, i.e, infeasible to find $\ddot{p}'$ `s.t.` $\texttt{ver}(h, \ddot{p}') = 1$, where $\ddot{p} \neq \ddot{p}'$.

22.2.1 A simple commitment scheme

- $\texttt{com}(m, d) : H(m \| d) = h$
- $\texttt{ver}(h, \ddot{p}) : H(m \| d) \stackrel{?}{=} h$ where $\ddot{p} = (m, d)$

22.3 Secret Sharing (brief recap)

Allows a secret to e split into multiple shares. Each share not revealing any information about the secret. If one group has enough shares, they can construct and learn the secret.

(t, n)-secret sharing schemes split the secret into n shares, `s.t.` t of the shares reveal no information about the original secret, while $t + 1$ shares allow complete reconstruction of it.

One reason SS is used in MPC is that it allows a party to run a computation on a share of other parties private inputs without learning anything about their inputs. After the shares are combined and the result is extracted, then the parties would learn only the result. Hence the inputs' privacy would be preserved.

22.4 Trusted Execution Environments (TEEs)

TEEs are secure processing environments (secure enclaves) that consist of processing, memory, and storage hardware units. In these environments, the residing code and data are isolated from other layers in the software stack including the operating system.

TEEs ensure *integrity* and *confidentiality* of data residing in them, and *correctness* of the computation delegated to them.

This is of course assuming that the physical CPU is not breached, enclaves are protected from an attacker even having physical access to the machine including the memory and the system bus.

TEEs are used in MPC because they offer input privacy and computational correctness, thus parties offload some part of the computation to TEEs in a privacy preserving manner. Since TEEs are susceptible to "side-channel" attacks, care should be taken when integrated into MPC protocols to ensure parties inputs are not leaked. TEEs can incur much lower overheads for MPC protocols that use them.

22.5 Blockchain Technology

MPCs can use a blockchain to design a "deposit scheme" whereby parties deposit some coins on the blockchain before the MPC protocol starts. During (or after) the execution of the protocol, if any party misbehaves, the deposits get taken by other parties in the protocol, therefore misbehaviour is penalised, incentivising honest behaviour.

23 Private Set Intersection (PSI)

PSI is a cryptographic protocol whose main goal is to compute the **intersection** of sets in a *privacy-preserving manner*, or to allow *sharing data* in a privacy-preserving way.

Ignoring privacy, the below diagram is a "set intersection" of Sets $A = \{1, 2, 3\}, B = \{1, 3, 4, 5, 9\}$.

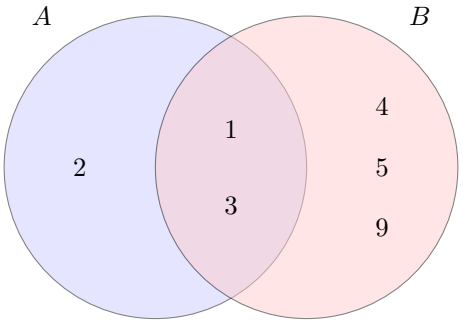


Figure 4: Set Intersection of $A \cap B = \{1, 3\}$.

PSI lets mutually distrustful parties compute the intersection of their private sets that nothing about the sets' elements beyond the result is revealed. In the example above, A must not learn anything about B's set beyond $A \cap B = \{1, 3\}$ (A cannot learn $\{4, 5, 9\}$) and vice-versa (B cannot learn $\{2\}$).

23.1 Adversary types of PSI

Recall that in general there are two types of adversaries:

- Passive/semi-honest
- Active/malicious

A PSI secure against **passive** adversaries need only satisfy *privacy*. However, to be secure against **active** adversaries, they must also ensure *correctness*.

A PSI (like other MPCs) is usually an interactive protocol where parties interact with each other, possibly over multiple rounds. Therefore, a PSI's design can be complicated when *efficiency* matters as well.

The problem is not trivial, since we also assume that a TTP does not exist for the parties.

23.2 Polynomial Representation

The idea of using a polynomial to represent a set' s elements was proposed by Freedman et al. In this representation, set elements $S = \{s_1, \ldots, s_d\}$ are defined over $\mathbb{F}$ and set S is represented as a polynomial of form:

$$\mathbf{p}(x) = \prod_{i=1}^{d}(x - s_i)$$

where $\mathbf{p}(x) \in \mathbb{F}[X]$ and $\mathbb{F}[X]$ is a polynomial ring.

Therefore, polynomial $\mathbf{p}(x) = \prod_{i=1}^{d}(x - s_i)$ represents set $S = \{s_1, \ldots, s_d\}$ because the polynomial's roots are the set elements.

Given $\mathbf{p}(x)$, we can find the set elements by finding the roots of the polynomial.

23.2.1 An example

If we add two polynomials, the result contains the intersection of the two polynomials' roots.

Example: assume we have two polynomials

$$\begin{cases} p_A(x) = (x-1)(x-2) \\ p_B(x) = (x-1)(x-5) \end{cases}$$

- Their sum is: $\phi = p_A(x) + p_B(x) = (x-1)(x-2) + (x-1)(x-5)$
- Value 1 is the root of the result, i.e., $\phi(1) = 0$
- But values 2 and 5 are not the roots of the result, i.e.,$\phi(5) \neq 0 \quad$ and $\quad \phi(2) \neq 0$

In this example: the "Greatest Common Divisor" (GCD) of $p_A(x)$ and $p_B(x)$ is polynomial $(x-1)$ that *contains the intersection of the two polynomials' roots.*

In general, the GCD of two polynomials is a polynomial that contains the intersection of the two polynomials' roots.

23.2.2 Features of polynomial representation

For two degree-d polynomials $\mathbf{p}_a$ and $\mathbf{p}_b$ (representing sets $S^{(A)}$ and $S^{(A)}$ resp.), and two degree-d random polynomials γ_A and γ_B whose coefficients are picked uniformly at random from the field, it is proven that $\theta = \gamma_A \cdot \mathbf{p}_A + \gamma_B \cdot \mathbf{p}_B = \mu \cdot \gcd(\mathbf{p}_A, \mathbf{p}_B)$, where μ is a uniformly random polynomial, and polynomial θ contains only information about the elements in $S^{(A)} \cap S^{(B)}$, and contains no information about other elements in $S^{(A)}$ or $S^{(B)}$.

Assuming the adversary knows one of the polynomials that represent the sets, e.g, it knows $\mathbf{p}_B$. This implies that given θ, if the adversary does not know polynomials γ_A and γ_B it cannot learn anything about $\mathbf{p}_A$.

23.3 Hash Functions

Hash functions produce a digest(fingerprint) of the input, taking a value of arbitrary size and returning a fixed-size output. They are one-way functions practically infeasible to invert. formally;

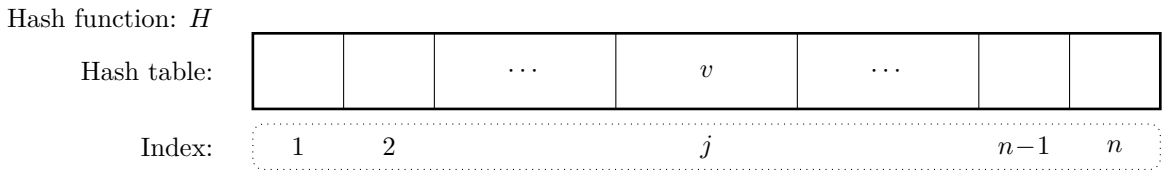
$$H : \{0,1\}^* \to \{0,1\}^\lambda$$

23.4 Pseudorandom Functions

Informally, they are a deterministic function that takes as input a key and some argument, outputting a value indistinguishable from that of a truly random function with the same domain and range.

23.5 Hash Tables

Accompanied by a hash function, they are arrays of bins, each of which can hold a set of elements. To insert an element, we compute it's hash and then store it in the bin whose index is the element's hash.



To insert value v into the table:

1. hash the value $H(v) = j$
2. insert v into the j-th cell of the table.

Hash tables are used in PSI because it allows set elements to be split into smaller parts (bins). Computation can then be performed on each cell (that contains a smaller number of set elements than the original set), allowing for performance improvements.

24 PSI Lab

Based on Kissner and Song (K&S) PSI.

We have two parties A (Alice) and B (Bob), in a semi-honest adversarial model. Relying on "threshold additive homomorphic encryption".

- **Addition:** $\text{Enc}(a) \oplus \text{Enc}(b) = \text{Enc}(a+b)$
- **Product** (of ciphertext $\text{Enc}(a)$ and plaintext b)**:** $\text{Enc}(a) \otimes b = \text{Enc}(a \cdot b)$

Allows no single-party decryption of ciphertext without counterparty collaboration.

24.1 Overview

1. Assume A has set $\{s_1^{(A)}, \ldots, s_n^{(A)}\}$ and B has $\{s_1^{(B)}, \ldots, s_n^{(B)}\}$.
2. Each party $I \in \{A, B\}$ represents their set as a polynomial

$$P^{(I)} = \prod_{i=1}^{n}(x - s_i^{(I)}) = x^n + b^{(I)}x^{n-1} + \ldots + c^{(I)}$$

3. Each party I uses a threshold additive homomorphic encryption (e.g, Paillier) to encrypt every coefficient of its polynomial:

$$\text{Enc}(P^{(I)}) = \text{Enc}(1), \text{Enc}(b^{(I)}), \ldots, \text{Enc}(c^{(I)})$$

4. Each party I sends their $\text{Enc}(P^{(I)})$ to the other party.
5. Each party I picks a random polynomial:

$$R^{(I)} = r_n^{(I)}x^n + r_{n-1}^{(I)}x^{n-1} + \ldots + r_0^{(I)}$$

6. Each party I homomorphically multiplies $R^{(I)}$ by the encryption of the polynomial it received.

$$\begin{cases} E^{(A)} = \text{Enc}(P^{(B)}) \otimes R^{(A)} = \text{Enc}(P^{(B)} \cdot R^{(A)}) \\ E^{(B)} = \text{Enc}(P^{(A)}) \otimes R^{(B)} = \text{Enc}(P^{(A)} \cdot R^{(B)}) \end{cases}$$

7. Alice sends $E^{(A)}$ to Bob, and Bob sends $E^{(B)}$ to Alice.
8. Each party computes

$$E = E^{(A)} \oplus E^{(B)} = \text{Enc}(P^{(B)} \cdot R^{(A)} + P^{(A)} \cdot R^{(B)})$$

9. They jointly decrypt E resulting: $D = P^{(B)} \cdot R^{(A)} + P^{(A)} \cdot R^{(B)}$
10. Each party evaluates D at each element of its set (e.g, $s_1^{(A)}$) and considers the element in the intersection if the evaluation is 0 (e.g,$D(s_1^{(A)}) = 0$)

Selective Disclosure

Privacy Enhancing Technologies (COMP0061)

Steven Murdoch | Sem 2 25/26 | 16-20 Feb

---

25 Access Control and User Privacy

Access Control systems are built on identity but this often reveals more information about a user than necessary. e.g; Bars require you to be older than 18, but the barman does not need to know your address, when checking your ID.

A privacy-invasive credential associates attributes with an identity. The usual way around this is to offer no privacy or soft privacy. e.g, Having an identity provider (such as apple or google) certify attributes about you. Matching credentials with a live person through biometrics.

26 Selective Disclosure Credentials

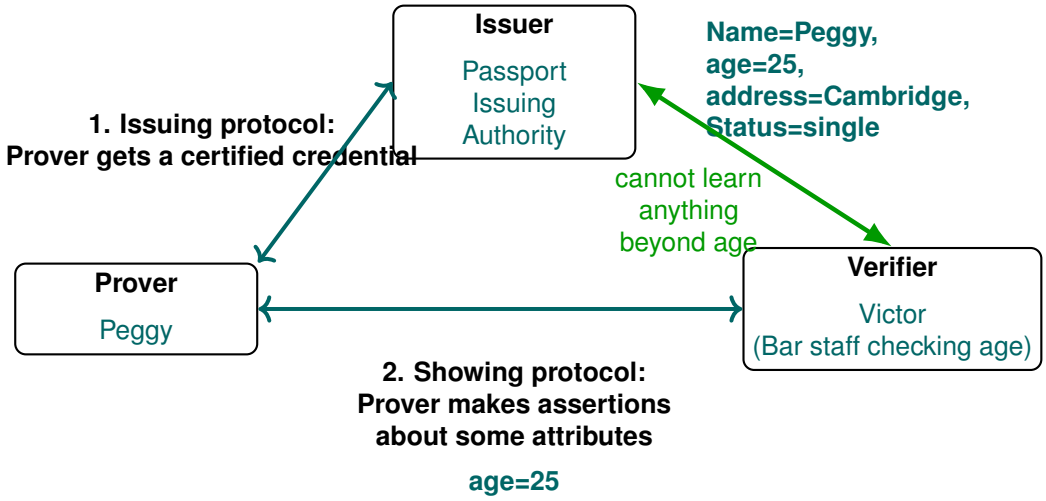
Selective Disclosure is about disclosing the minimum amount of information about you, needed to grant access. The participants of s SD system include:

- **Issuer**: The identity provider
- **Prover**: The subject
- **Verifier**: The relying party

The properties we want of this system are:

- The prover convinces the verifier that he holds a credential with attributes that satisfy some boolean formula.
- The prover cannot lie.
- The verifier does not infer anything else aside the formula.
- Anonymity is maintained even if the verifier and issuer collude.

As illustrated:



26.1 Families of Selective Disclosure Protocols

Selective disclosure credentials are available from three families, each with their strengths. Their differences are discussed in Table 1.

Single-show credential

- Brands & Chaum
- Blind the issuing protocol
- Show the credential in clear
- Multiple shows are linkable - *requires re-issuing*
- Used by Microsoft U-Prove (defunct)

Multi-show

- Camenisch & Lysyanskaya
- Blinded showing - Prover shows that they know a signature over a particular ciphertext.
- Cannot link multiple shows of the credential
- More complex - *tricky to implement*
- Used by IRMA

Algebraic Message Authentication Codes

- Meiklejohn et al.
- Restriction: Issuer is Verifier, or on-line check – special case
- Blinded proof of validity of MAC
- Fast and elegant!

Algebraic MACs require a few cryptographic primitives including: the DLP (Section 13), Schnorr's ID Protocol (Section 18.2), Proof of DL and linear relations of attributes (Section 20).

27 SD Building Blocks

27.1 Discrete Logarithm Problem

The DLP is hard given that knowing g and $g^x \pmod p$, it is difficult to find $x = \log_g g^x \pmod p$. But exponentiating g to the power x as $g^x \pmod p$ is easy to compute.

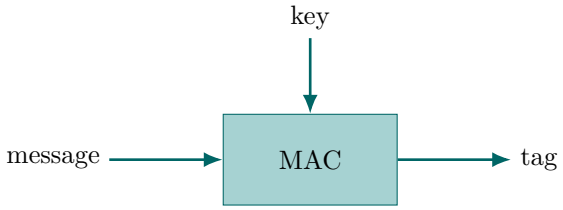
Similar to the Diffie Hellman problem, given (g, g^x, g^y) it is difficult to find g^{xy}. [1]

27.2 Schnorr's Identification Protocol (recap)

SD is a fairly good example of a ZK use case. Schnorr's ID protocol aims to protect a private x, given public parameters $g, y = g^x$. A prover is able to show they know x without revealing it, and can be made non-interactive with Fiat-Shamir (hashing).

27.3 Message Authentication Codes (MACs)

MACs are traditionally a symmetric cryptography primitive, and have two inputs (message and key), and one output (tag).



To an adversary without the secret key, MAC is computationally indistinguishable from a pseudorandom function. This ensures unforgeability, meaning an attacker cannot alter a message or create a new one without being detected.

This provides data origin authentication and integrity between Alice and Bob, commonly used to prevent users from tampering with the data stored in HTTP cookies.

27.4 Building SD with MACs

Let's imagine for a second that the Issuer and the Verifier are the same person, and we simply just want to use a MAC to protect integrity but not privacy.

As the prover, we request and retrieve certification for our credential (C) through a MAC protocol ($\text{MAC}_K(C)$) from the Issuer, who has the secret key K. We can then prove our ownership of the tagged credential by *showing* it to the verifier (also the issuer).

This obviously has no privacy, however, we can now add ZK to the *showing* protocol through an anonymous channel, and prove that we have C with some attribute and a valid MAC on them.

Some problems with this:

[1] This is a stronger assumption than the DLP.

Feature	**Brands–Chaum / Single-show**	**Camenisch–Lysyanskaya / Multi-show**	**Algebraic MAC-based**
Reusability	Single-use; repeated shows are linkable.	Reusable; multiple shows are unlinkable.	Reusable in restricted settings.
Showing protocol	Credential/token is shown mostly in clear.	Prover gives a blinded proof of knowledge of a valid signature.	Prover gives a blinded proof of validity of a MAC.
Unlinkability	Requires re-issuance for unlinkable repeated use.	Built-in unlinkability across presentations.	Cryptographically private, but online checks may leak metadata.
relation of issuer-verifier	Issuer and verifier may differ.	Issuer and verifier may differ; public verification is possible.	Usually issuer is verifier, or verifier needs online issuer check.
Complexity	Simple and efficient.	More complex and harder to implement.	Fast and elegant, but deployment assumptions are restrictive.
Best use case	One-time tickets, vouchers, or access tokens.	Reusable privacy-preserving identity credentials.	Closed ecosystems where online verification is acceptable.

Table 1: Key differences between selective disclosure credential families.

- The prover needs to make sure all credentials use the same MAC key, otherwise we can infer provers through identifying which of them use which MAC key.
- The anonymous channel is a requirement to prevent identification of transactions.
- The prover does not know the secret key K.

28 Algebraic MACs (AMACs)

Invented by Sarah Meiklejohn et al. (2014), AMACs have some useful properties for ZKPs.

- They can efficiently prove correct MAC creation (issuing)
- They can efficiently prove correct MAC possession (showing)

Key generation for AMACs have the same public parameters and assumptions as ordinary MACs. Public g, h, with group $\mathbb{G}$ of prime order p, and assume DL and DH are hard.
secrets are encoded as $\text{sk} = \{x_0, x_1, \ldots, x_k\}$ where k is the number of attributes to be encoded.
The public issuer parameters are used to facilitate proofs later: publishing

$$\text{iparams} = \{X_i = h^{X_i} \text{ for } i > 0\}$$

AMAC generation should create (u, u') for a message m. Taking inputs:

$$\text{sk} = \{x_0, x_1, \ldots, x_k\}$$

and

$$m = \{m_i\} \text{ for } 1 \leq i \leq k$$

The MAC tag generation is as follows:

- Generate random $u \overset{\$}{\in} \mathbb{G}/\{1\}$
- Compute $u' = u^{H_{\text{sk}}(m)}$ where $H_{\text{sk}}(m) = x_0 + \sum m_i x_i$.
- Output tag $= (u, u')$.

MAC verification uses u, u', sk, m. Given (u, u'), the verifier checks $u' = u^{H_{\text{sk}}(m)}$ and determines whether it is a valid MAC for $m = \{m_i\}$. This reveals u, u' and m_i so is not private.

29 Implementing Selective Disclosure

We can add ZK to the AMAC credential issuing system above to implement a privacy preserving selective disclosure algorithm.

Our setup algorithm must output $(\mathbb{G}, p, g, h)$, same as above. The credential key generation is somewhat modified.

$$\begin{aligned} &\text{CredKeyGen}() : \\ &(\text{KeyGen}_{\text{AMAC}}() \rightarrow \text{sk} = \{x_0, \ldots, x_k\}, \text{iparam} = \{X_1, \ldots, X_k\}), \\ &o_{xo} \overset{\$}{\leftarrow} \mathbb{Z}_p, \\ &C_{xo} = g^{x_0} h^{o_x o} \end{aligned}$$

Which outputs public $(\text{iparam}, C_x o)$ and private $(sk, o_x o)$.
Our Issuer computes:

$$\begin{aligned} \text{Tag}(u, u') &= \text{AMAC}(sk, \{m_i\}), \\ \text{Proof } \pi_0 &= \text{NIZK}\{(sk = \{x_i\}, o_{xo}) : \\ u' &= u^{x_o} \sum (u^{m_i})^{x_i} \text{ and} \\ C_{xo} &= g^{x_0} h^{o_{xo}} \text{ and} \\ X_i &= h^{X_i} \text{ for } 0 < i < k + 1\} \end{aligned}$$

and returns as output: $(u, u'), \pi_0$.

The Prover (Peggy) must convince the Verifier that she holds such a credential. Peggy generates:

- Random $a \xleftarrow{\$} \mathbb{Z}_p$ and $u_a, u'_a = (u^a, u'^a)$
- Random $z_i \xleftarrow{\$} \mathbb{Z}_p$ and $C_{m_i} = u^{m_i} h^{z_i}$
- Random $r \xleftarrow{\$} \mathbb{Z}_p$ and $C_{u'} = u'_a g^r$
- Prove:

$$\pi_1 = \text{NIZK}\{(z_i, r, m_i) : \\ C_{m_i} = u^{m_i} h^{z_i} \\ V = g^{-r} \sum X_i^{z_i} + \ldots\}$$

- Output $(u_a, C_{m_i}, C_{u'}), \pi_1$

The Verifier/Issuer (Victor) computes:

- $V = u_a^{x_0} \sum C_{m_i}^{x_i} / C_{u'}$
- Then Verify π_1 using V.

However, the issuer is not necessarily going to always be the verifier. In that case we have two options: (1) use Camenisch-Lysianskaya or Brands-Chaum. (2) Use MACs + online check.

30 Applications of Selective Disclosure

Anonymous credentials can not only be used for privacy-preserving access control, but also for privacy-preserving identity management.

A service may provide you with an identity to show to other services for logging in, SD allows the service to encode attributes of a credential and prove your right to a service to third parties, subject to AMACs requiring online checks.

We may extend this usage to Identity Cards with privacy friendly e-identities.

Data and Query Anonymization

Privacy Enhancing Technologies (COMP0061)

Steven Murdoch | Sem 2 25/26 | 23-27 Feb

31 Why Have Anonymous Data?

Data anonymization serves as a critical strategic tool for organizations seeking to balance data utility with privacy obligations. By stripping identifying characteristics from raw datasets, entities can facilitate internal data sharing while mitigating the risk of accidental loss and share information with external partners without compromising proprietary interests. Furthermore, robust anonymization can exempt datasets from stringent regulatory frameworks like the General Data Protection Regulation (GDPR), which do not apply to data that can no longer be linked to a natural person. This is particularly vital for longitudinal research, such as medical studies or national census reporting, where the objective is to extract statistical insights or provide interactive query services without exposing individual identities.

There are historical precedents that demonstrate simple de-identification—the removal of direct identifiers like names or social security numbers, as fundamentally insufficient for true anonymization. High-profile failures, such as the 2006 AOL search data leak and the 2007 Netflix Prize dataset release, illustrate that pseudonymized data remains vulnerable to re-identification. In both instances, researchers and journalists were able to cross-reference seemingly anonymous records with external datasets, such as public IMDB profiles or specific search patterns, to unmask individuals. These cases prove that pseudonymization is not a robust anonymization method, as the unique behavioral "fingerprints" left within the data often provide enough context to reverse the process.

31.1 k-anonymity

Say we want to protect a database table with "k-anonymity". We assume 1 person per row, delete all personal identifiers, identify all quasi-identifiers, and ensure at least k entries share the same set of quasi-identifiers through suppression and generalization.

Notably, there is an efficiency problem with this as suppressing everything is k-anonymous but provides no utility. It is fairly expensive to find a k-anonymity transformation that preserves maximal utility.

Two security problems with k-anonymity:

- **Problem 1**: If other data has little diversity, information can be leaked through homogeneity attacks.
- **Problem 2**: In the presence of side-information, any information becomes a quasi-identifier (background knowledge attack).

31.2 l-diversity

A q-block is l-diverse if contains at least l "well represented" values for the sensitive attribute S. A table is l-diverse if every q-block is l-diverse.

e.g, If each q-block has at least three values for the sensitive attribute, it is 3-diverse.

This however does not consider the semantic closeness of attributes. e.g, Bob lives in zip-code 47678 and is 27 years old.

From the table, we would be able to infer that Bob earns $20 - 40$K and has a stomach related disease.

31.3 t-closeness

Therefore we want some way of extending l-diversity by taking into account distribution of values. An equivalence class is said to have t-closeness if the distance between the distribution of a sensitive attribute in this class and the distribution of the attribute in the whole table is no more than a threshold t. A table is said to have t-closeness if all equivalence classes have t-closeness. In other words, the statistical profile of the sensitive attribute in a specific group, must be very similar to the overall population's profile.

Zip code	Age	Salary	Disease
476***	2*	20K	Gastric Ulcer
476***	2*	30K	Gastritis
476***	2*	40K	Stomach Cancer
4790*	>40	50K	Gastritis
4790*	>40	100K	Flu
4790*	>40	70K	Bronchitis
476**	3*	60K	Bronchitis
476**	3*	80K	Pneumonia
476**	3*	90K	Stomach Cancer

It is rare that releasing data sets after sanitization can preserve both privacy and utility, particularly against adversaries with side information. Therefore, we generally only release rich "anonymized" datasets under licence to "trusted" parties.

32 Personally Identifiable Information (PII)

The concept of PII is to separate identifiers from other data items. As any information about an individual can be used to identify them within an anonymized dataset.

Any attribute or piece of data that, when viewed individually or combined with other available data (quasi-identifiers, structural knowledge, side-information), allows an adversary to distinguish or identify a specific individual within a dataset.

Social network graphs don't really align well with table-based anonymization systems. It is very hard to anonymize "rich" graph data. Attacks may deploy structural or seed based attacks to link nodes. An attacker is very easily able to infer a user, just from side-information alone.

33 Anonymizing Statistics

Raw data allows any attacker with a dataset to deanonymize users fairly easily. But if we only need the data for statistical analysis, then we don't need to store accurate information to derive statistical knowledge.

33.1 Differential Privacy

A safer metric of anonymization, Differential privacy protects against side-information, should compose nicely if more information is released, and would provide the adversary no more information about an individual than they would learn if the individual was not in the dataset.

There is likely not a much stronger definition of privacy than this, if we consider the goal of "learning nothing about an individual.", when we consider the case where you learn some statistic on a given population, you always gain information about any and all users.

Differential privacy means that an individual's contribution to the dataset is negligible such that when we consider two databases $\mathrm{DB}_1, \mathrm{DB}_2$, the probability of getting any statistic $K = f(\mathrm{DB}_i)$ should be very close (ϵ) in both databases.

$$\forall K, (\mathrm{DB}_1, \mathrm{DB}_2) : \frac{\Pr[K|\mathrm{DB}_1]}{\Pr[K|\mathrm{DB}_2]} < e^{\epsilon}$$

Where $e^{\epsilon} \approx 1 + \epsilon$, for very small ϵ, therefore, a smaller ϵ leads to better privacy.

Generally for all databases $\mathrm{DB}_1, \mathrm{DB}_2 \in \{$ set of all related databases$\}$, and all statistics outputs K.

$$\Pr[K|\mathrm{DB}_1] < (1+\epsilon)\Pr[K|\mathrm{DB}_2] < e^{\epsilon}\Pr[K|\mathrm{DB}_2]$$

DP deals with probability of a statistic result K given a database ($\Pr[K|\mathrm{DB}]$), do we have to worry about what we can infer about DB given the observed statistic? No, the implication of the bound on $\frac{\Pr[K|\mathrm{DB}_1]}{\Pr[K|\mathrm{DB}_2]} < e^{\epsilon}$ implies a bound on posteriors

$$\frac{\frac{\Pr[K|\mathrm{DB}_1]\Pr[\mathrm{DB}_1]}{\Pr[K]}}{\frac{\Pr[K|\mathrm{DB}_2]\Pr[\mathrm{DB}_2]}{\Pr[K]}} < e^{\epsilon}\frac{\Pr[\mathrm{DB}_1]}{\Pr[\mathrm{DB}_2]} \implies \frac{\Pr[\mathrm{DB}_1|K]}{\Pr[\mathrm{DB}_2|K]} < e^{\epsilon}\frac{\Pr[\mathrm{DB}_1]}{\Pr[\mathrm{DB}_2]}$$

Consider two databases with only one element being different. One can define a differentially private mechanism if for all observations, and pairs of databases, we have:

$$\frac{\Pr[\text{obs}|\text{DB} : r_i = a]}{\Pr[\text{obs}|\text{DB} : r_i = b]} < e^{\epsilon}$$

Consider observing K_1 and K_2, that are both ϵ-differentially private. This is also differentially private:

$$\frac{\Pr[K_1, K_2|\text{DB}_1]}{\Pr[K_1, K_2|\text{DB}_2]} < e^{2\epsilon}$$

Notice the more statistics you release, the larger ϵ is, and the lower privacy protection we have. We must therefore decide on an upper bound on ϵ and stop answering queries after it has been reached.

We may decide how "sensitive" we want f to be on the inclusion or exclusion of a record by using a Laplace mechanism to add random noise. Adding a distribution $f(x|b) = \frac{1}{2b}e^{-\frac{|x|}{b}}$ with mean 0 and scale parameter b. $K(\text{DB}) = f(\text{DB}) + \text{Laplace}(0, b)$ is differentially private with $\epsilon = \Delta\frac{f}{b}$.

DP noise harms the utility of the data, but that's often fine. For a small number of participants, noise overwhelms statistics, but for larger N, noise becomes insignificant.

Privacy in Data Storage and Retrieval

Privacy Enhancing Technologies (COMP0061)

Steven Murdoch | Sem 2 25/26 | 2-6 Mar

---

34 Private Storage

Local storage is fairly simple to protect, really only including the threat of stolen hardware or being compelled to reveal your passwords and encryption keys.

Remote storage is under threat of a third party reading your data, or being monitored for patterns of access and revealing the nature of the data. Remote storage can also be suppressed and DoS'd.

There are also efficiency considerations, is the hardware fast random access? can you update your model quickly? or search and select records by content?

If you want to protect your data directly, this can be a difficult task, encryption of bulk storage imposes challenges not faced for message encryption.

- Disk block level (may not be a multiple of block cipher size)
- Random access reading
- Random access writing
- No extra space for an IV of a MAC

Ideas: use block ID as seed for IV, use a really large block cipher, and have file system level protections.

34.1 Bitlocker

Was a windows feature intended to protect against a lost laptop or hard drive. The idea is to use AES-CBC to encrypt each block, then decrypt whole blocks in one go. The IV can be derived from the disk block number using the key.

For integrity, since there is no room for a MAC tag per block, in case of ciphertext modification, the whole block becomes garbage. This way we avoid subtle errors by destroying all semantics.

Key management is done through the computer's Trusted Platform Module, and may allow for key escrow within an organisation for recovery.

35 Deniability

We may want deniability for our storage system in the case of coercion of the user.

Deniability and Forward secrecy is hard to achieve, the intrinsic limit is that the user needs to recover encrypted data, therefore may be forced to do so under coercion. Some mechanisms that can be used include;

- Using multiple levels of ciphertexts - forward-secure steganography
- Hiding messages in cover - multimedia steganography
- Forward secrecy and append only storage

35.1 Steganography

Steganography is the act of embedding secret messages within cover media (images, audio recordings, videos). We can further use regions of "randomness" to make the new image inconspicuous.

The secret key chooses the locations to embed and encrypt text, resulting in plausible deniability of present content.

35.2 Append only storage

Use an input key K_0 and store it offline somewhere safe. Append only storage prevents old data from being recovered as the system automatically encrypts each entry m_i with key K_i and updates the key automatically (potentially with a hash function $K_{i+1} = H(K_i)$), and deleting K_i.

36 Private Remote Storage

To protect against a 'malicious' provider or network-based attackers, we need a different method of storage and retrieval. Take the now defunct example of Tahoe-LAFS;

- Confidentiality relies on client side components
- Cryptography is used to protect data
- The servers are not 'trusted'
- Multiple servers are used for high availability

Remote storage allows us to store and retrieve blocks on a remote server and protects against eavesdroppers and corrupt services.

36.1 ORAM

Secure client-side cloud storage is designed to provide confidentiality, integrity and availability of accessing content. However, we still do not protect against traffic analysis of block reads and writes. These patterns of access may leak information about activities or the content of those blocks.

Therefore, Oblivious Random Access Memory is used by limiting local storage to hide patterns of access to larger remote storage. This may require modifying the protocol for writing blocks, and in practice, there are more efficient protocols.

The idea is for the client to store a small number of items locally (in a shelter). Initialisation is handled by storing a randomly permuted sequence of the blocks $\pi(i)$ on server. To access block i, the client checks whether it is in the local shelter, then;

- If yes: read and write a random new position on the server and place it in the shelter.
- If no: read and write position $\pi(i)$ and store item i in the shelter.

The reason for doing so is that every access i leads to a statistically independent access j. When the shelter is full, obliviously permute a fresh remote database (a hard problem).

After oblivious shuffle, the server cannot know which new block matches each old block. To implement an oblivious shuffle, with limited local storage, we can use an oblivious sorting network.

37 Private Information Retrieval

If we wanted to access a public database record without allowing the server to know which record you are accessing, is this possible? trivially, yes, by downloading the whole database. More practical PIR protocols must be more efficient, but must still hide access patterns.

PIR systems can be built around an additively homomorphic encryption system. Consider the database of n records. Shaping the database into a square matrix M of size $\sqrt{n} \times \sqrt{n}$. To query i:

- Identify row j holding record i and generate a key for an additively homomorphic encryption scheme (i.e, Paillier).
- Make a vector v of ciphertexts, one for each row: $E(0)$ for all rows except the target row j.
- Send the row to the server that computes and returns $v \cdot M$, and decrypt the returned row and retrieve the record.

This Communication costs $2 \cdot \sqrt{n}$ and requires n computations.

37.1 Multi-Server PIR

Using PIR for multiple servers requires secret sharing rather than homomorphic encryption for efficiency, since the cost of single server PIRs are too high for many scenarios.

Assume k servers include at least 1 honest server. We use secret sharing for encryption of 0 or 1 as $<0>$ or $<1>$, and $m \cdot <0> = <0>$ and $m \cdot <1> = <m>$. Each server combines the shared query with the database and returns the resulting shares.

Communication now costs $2 \cdot k \cdot \sqrt{n}$ and computation is $O(n)$. Practically, the bottleneck is in the memory fetch time which can be parallelised.

37.2 Private keyword search

Say Alice stores her emails on a remote server and wants all emails containing a specific keyword. A simple solution is to build a keyword-to-email index, download the whole index, search locally, and retrieve only the matching emails. This leaks;

- How many emails were retrieved.
- Which email IDs were retrieved.
- Whether later searches retrieve the same email set.

Encryption-based search can reduce cost by allowing the server to look up encrypted keywords directly, where the same keyword always encrypts to the same ciphertext.

Order-preserving encryption preserves order:

$$m_0 < m_1 \Rightarrow E(m_0) < E(m_1)$$

These methods are weaker than randomized encryption because they leak equality or order information. A typical approach is to encrypt index terms with deterministic encryption, then query with the encrypted keyword and retrieve the encrypted index entry and the matched entries.

We can apply search techniques to streaming data. High-volume data streams, private search can use additively homomorphic encryption.

Alice encrypts a keyword-interest map where each keyword has a hidden bit:

$$l_i \in \{0, 1\}$$

The server uses this encrypted map to test documents against Alice's hidden keywords. It places encrypted match results into a buffer and sends the buffer back. Alice decrypts the buffer, resolves collisions, and recovers the matching documents.

The goal is to let the server filter the stream without learning Alice's keywords.