CS 458 / 658: Computer Security and Privacy

Module 5 – Security and Privacy of Internet Applications

Part 1 – Basics of Cryptography

Fall 2022

Module outline

1. Introduction to cryptography
2. Secret-key Cryptography
3. Public-key Cryptography
4. Integrity
5. Authentication

Cryptology

- Cryptology is a science that studies:
  - Cryptography ("secret writing"): *Making secret messages*
    - Turning plaintext (an ordinary readable message) into ciphertext (secret messages that are "hard" to read)
  - Cryptanalysis: *Breaking secret messages*
    - Recovering the plaintext from the ciphertext
- The point of cryptography is to send secure messages over an insecure medium (like the Internet)
- Cryptanalysis studies cryptographic systems to look for weaknesses or leaks of information

The scope of these lectures

- The goal of the cryptography unit in this course is to show you what cryptographic tools exist, and information about using these tools in a secure manner
- We won't be showing you details of how the tools work
    - For that, see CO 487, chapter 2 of van Oorschot's textbook, or chapter 2.3 of Pfleeger's textbook

Cast of Characters

- When talking about cryptography, we often use a standard cast of characters

(Honest) communicating parties

Adversaries

Alice

Bob

Carol

Dave

Eve

Mallory

- Eve: a **passive** eavesdropper who can listen to transmitted messages
- Mallory: an **active** Man-In-The-Middle, who can listen to, and modify, insert, or delete, transmitted messages.
- There are others: Trent (a Trusted Third Party), Peggy (a prover), Victor (a verifier), etc.

Building blocks

- Cryptography contains three major types of components
  - Confidentiality components
    - Preventing Eve from reading Alice's messages
  - Integrity components
    - Preventing Mallory from modifying Alice's messages without being detected
  - Authenticity components
    - Preventing Mallory from impersonating Alice

Kerckhoff's principle

Kerckhoff's principle: a cryptosystem should be secure, even if everything about the system, except the key, is public knowledge.

Shannon's maxim: one ought to design systems under the assumption that the enemy will immediately gain full familiarity with them.

- So don't use "secret" encryption methods (security by obscurity)
- Have public algorithms that use a secret key as input
- It's easy to change the key; it's usually just a smallish number

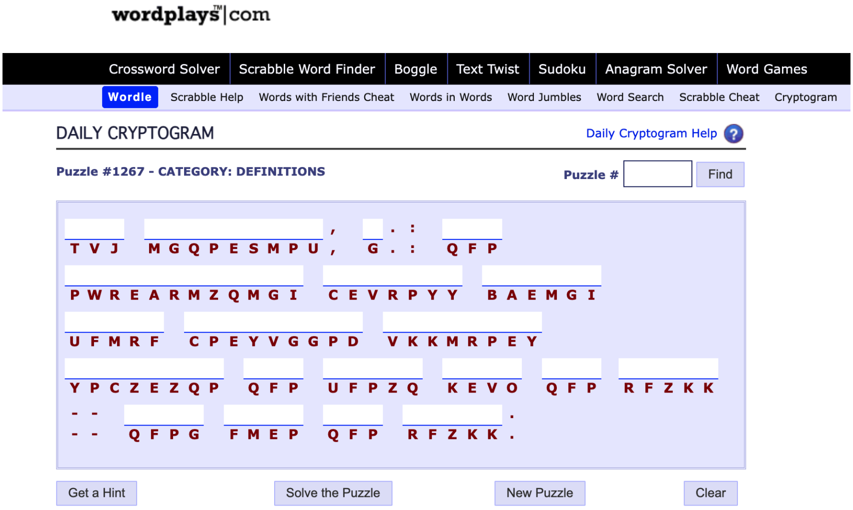
Kerckhoffs' principle

Kerckhoffs' principle has a number of implications:

- The system is *at most* as secure as the number of keys
- Eve can just try them all, until she finds the right one
- Many times, there are shortcuts to finding the key

- Example: newspaper cryptogram has 403,291,461,126,605,635,584,000,000 possible keys
- But you don't try them all; it's way easier than that!

Daily cryptogram



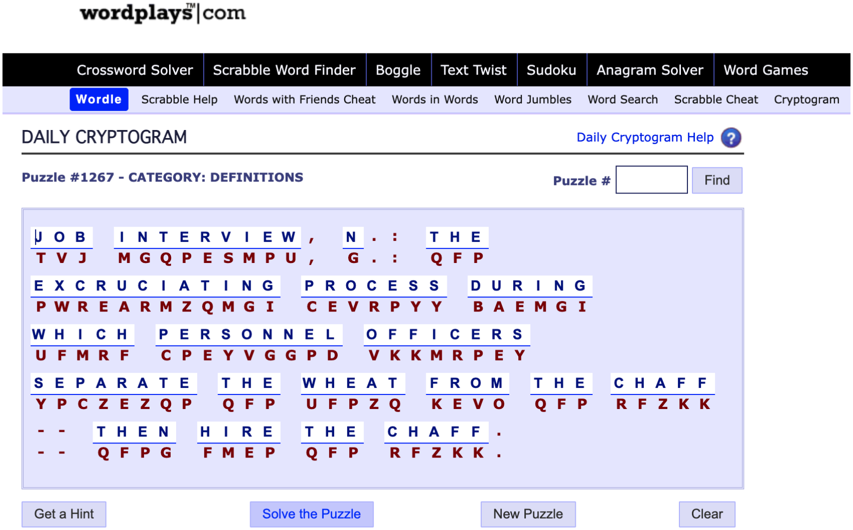
DAILY CRYPTOGRAM

Puzzle #1267 - CATEGORY: DEFINITIONS

Puzzle #

TVJ MGQPESMPU, G.: QFP
PWREARMZQMGI CEVRPYY BAEMGI
UFMRF CPEYVGGPD VKKMRPEY
YPCZEZQP QFP UFPZQ KEVO QFP RFZKK
-- QFPG FMEP QFP RFZKK.

Daily cryptogram



DAILY CRYPTOGRAM

Puzzle #1267 - CATEGORY: DEFINITIONS

Puzzle #

```
JOB INTERVIEW, N.: THE
TVJ MGQPESMPU, G.: QFP
EXCRUCIATING PROCESS DURING
PWREARMZQMGI CEVRPYY BAEMGI
WHICH PERSONNEL OFFICERS
UFMRF CPEYVGGPD VKKMRPEY
SEPARATE THE WHEAT FROM THE CHAFF
YPCZEZQP QFP UFPZQ KEVO QFP RFZKK
-- THEN HIRE THE CHAFF.
-- QFPG FMEP QFP RFZKK.
```

Strong cryptosystems

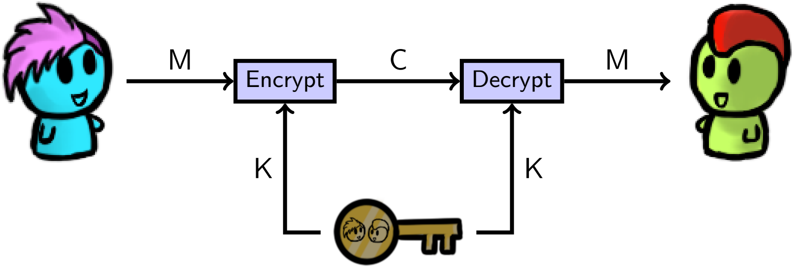
- What information do we assume the attacker (Eve) has when she's trying to break our system?
- She may:
  - Know the algorithm
  - Know some part of the plaintext
  - Know a number (maybe a large number) of corresponding plaintext/ciphertext pairs
  - Have access to an encryption and/or decryption oracle
- And we still want to prevent Eve from learning the key!

Module outline

1. Introduction to cryptography
2. Secret-key Cryptography
3. Public-key Cryptography
4. Integrity
5. Authentication

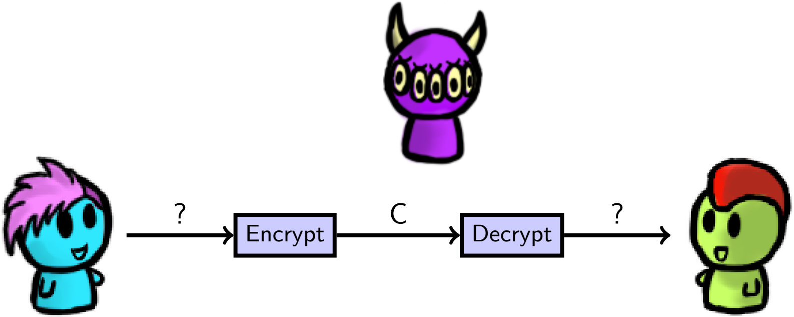
Secret-key encryption

- Secret-key encryption is the simplest form of cryptography
- Also called symmetric encryption
- Used for thousands of years
- The key Alice uses to encrypt the message is the same as the key Bob uses to decrypt it



Secret-key encryption

- Eve, not knowing the key, should not be able to recover the plaintext



Vernam Cipher

- Encrypts one bit at a time by XOR'ing the plaintext with the key:
- Plaintext (t bits): $M = [m_1, m_2, \ldots, m_t]$
- Key (t bits): $K = [k_1, k_2, \ldots, k_t]$
- Ciphertext (t bits): $C = [c_1, c_2, \ldots, c_t] = [m_1, m_2, \ldots, m_t] \oplus [k_1, k_2, \ldots, k_t]$
- XOR reminder:

$$0 \oplus 0 = 0 \qquad 0 \oplus 1 = 1 \qquad 1 \oplus 0 = 1 \qquad 1 \oplus 1 = 0$$

Q: How do we decrypt?

A: $[m_1, m_2, \ldots, m_t] = [c_1, c_2, \ldots, c_t] \oplus [k_1, k_2, \ldots, k_t]$

- If K is randomly chosen and *never reused*, Vernam cipher is called One-Time Pad

One-time Pad

- Vernam cipher: $C = M \oplus K$
- If K is randomly chosen and never reused, this is called One-Time Pad
- This provides Information-Theoretic security.

Q: Why does "try every key" not work here?

A: Because, given a ciphertext C, for every possible message M, there exist a key K that could have generated that ciphertext.

Q: Does this provide integrity?

One-time Pad

- Vernam cipher: $C = M \oplus K$
- If K is randomly chosen and never reused, this is called One-Time Pad

Q: If your boss stores your salary (in binary) encrypted with a one time pad, and you have access to the ciphertext, what can you do with it?

A: You can XOR a "100000 . . .". This flips the most significant bit, which most likely will be zero.

- The one-time pad is very hard to use correctly:
  - The key must be truly random, not pseudorandom
  - Keys would have to be shared in person, or sent by courier.
  - The key must never be used more than once!
    - A "two-time pad" is insecure!

Computational security

- In contrast to One-Time Pad's "perfect" or "information-theoretic" security, most cryptosystems have "computational" security.
- This means that it's certain they can be broken, given enough work by Eve
- How much is "enough"?
- At worst, Eve tries every key
  - How long that takes depends on how long the keys are
  - But it only takes this long if there are no "shortcuts"!

Trying every key: some data points

These are some estimates for RC5:

- One computer can try about 17 million keys per second: $1.7 \cdot 10^7$ keys/second.
- A medium-sized corporate or research lab may have 100 computers: $1.7 \cdot 10^9$ keys/second.
- The Bitcoin network computes 258 million terahashes per second as of Oct 2022. If the hardware could be used to try decrypting with a key in the same time, that's $\approx 2.6 \cdot 10^{20}$ keys/second.

40-bit crypto

- This was the US legal export limit for a long time (cryptosystems were classified as munitions until the late 90's)
- $2^{40} \approx 1 \cdot 10^{12}$ possible keys

Key size	Computer	Lab	Bitcoin network
key/second	$\approx 1.7 \cdot 10^7$	$\approx 1.7 \cdot 10^9$	$\approx 2.6 \cdot 10^{20}$
40-bit	18 hours	11 minutes	4.2 ns

56-bit crypto

- This was the US government standard (DES) for a long time
- $2^{56} \approx 7.2 \cdot 10^{26}$

Key size	Computer	Lab	Bitcoin network
key/second	$\approx 1.7 \cdot 10^7$	$\approx 1.7 \cdot 10^9$	$\approx 2.6 \cdot 10^{20}$
40-bit	18 hours	11 minutes	4.2 ns
56-bit	134 years	16 months	0.22 ms

128-bit crypto

- This is the modern standard
- $2^{128} \approx 3.4 \cdot 10^{38}$

Key size	Computer	Lab	Bitcoin network
key/second	$\approx 1.7 \cdot 10^7$	$\approx 1.7 \cdot 10^9$	$\approx 2.6 \cdot 10^{20}$
40-bit	18 hours	11 minutes	4.2 ns
56-bit	134 years	16 months	0.22 ms
128-bit	$6.3 \cdot 10^{23}$ years	$6.3 \cdot 10^{21}$ years	$4.1 \cdot 10^{10}$ years

- $4.1 \cdot 10^{10}$ years: around 3 times larger than the age of the universe, around 4.2 times larger than the expected lifetime of the sun.

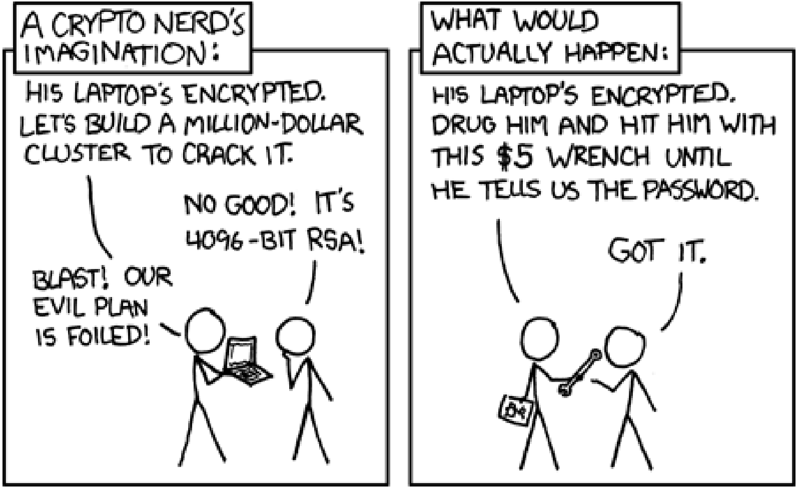
Well, we cheated a bit

- This isn't really true, since computers get faster over time
  - A better strategy for breaking 128-bit crypto is just to wait until computers get 2^{88} times faster, then break it on one computer in 18 hours.
  - How long do we wait? Moore's law says 132 years.
  - If we believe Moore's law will keep on working, we'll be able to break 128-bit crypto in 132 years (and 18 hours) :-)

Q: Do we believe this?

Q: What about quantum computers? (Shor's Algorithm)

An even better strategy

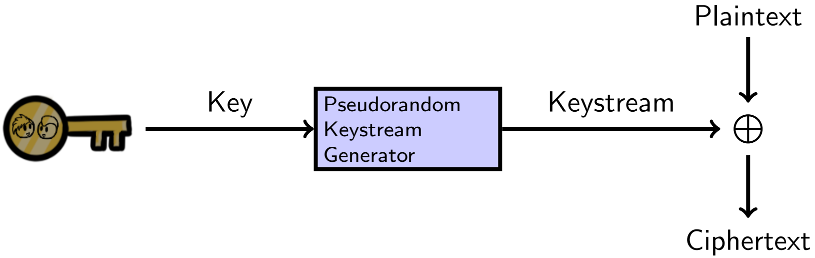

A CRYPTO NERD'S IMAGINATION:
HIS LAPTOP'S ENCRYPTED. LET'S BUILD A MILLION-DOLLAR CLUSTER TO CRACK IT.
NO GOOD! IT'S 4096-BIT RSA!
BLAST! OUR EVIL PLAN IS FOILED!
WHAT WOULD ACTUALLY HAPPEN:
HIS LAPTOP'S ENCRYPTED. DRUG HIM AND HIT HIM WITH THIS $5 WRENCH UNTIL HE TELLS US THE PASSWORD.
GOT IT.

Types of secret-key cryptosystems

- Secret-key cryptosystems come in two major classes
  - Stream ciphers
  - Block ciphers

Stream ciphers

- A stream cipher operates one bit at a time.
- Basically, take the One-Time Pad, but use a pseudorandom keystream instead of a truly random one



- RC4 was the most common stream cipher on the Internet but deprecated. ChaCha increasingly popular (Chrome and Android), and SNOW3G in mobile phone networks.

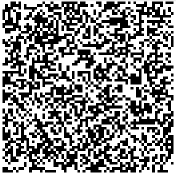
Issues of Two-time Pad

Q: What happens if you use the same key (therefore, same keystream) to encrypt two messages? $C_1 = M_1 \oplus K\,, \quad C_2 = M_2 \oplus K$

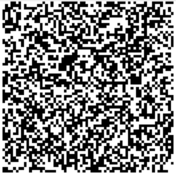
A: We can XOR the ciphertexts: $C_1 \oplus C_2 = (M_1 \oplus K) \oplus (M_1 \oplus K) = M_1 \oplus M_2$

Q: Why is this an issue?

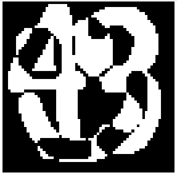
A: Messages are not purely random!



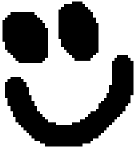
C_1



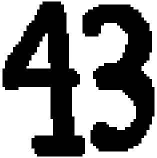
C_2



$C_1 \oplus C_2$



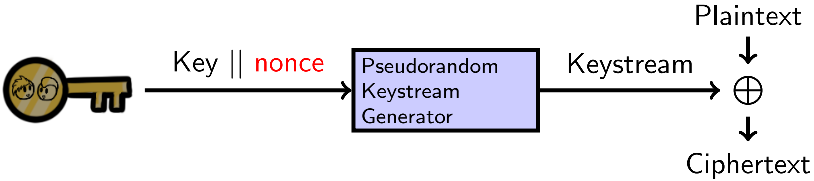
M_2



M_1

Stream ciphers

- Stream ciphers can be very fast
  - This is useful if you need to send a lot of data securely
- But they can be tricky to use correctly!
  - We saw the issues of re-using a key! (two-time pad)
  - Solution: concatenate key with nonce (we'll see more about nonces later)



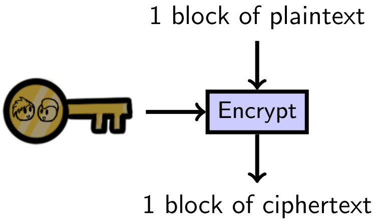
- WEP, PPTP are great examples of how not to use stream ciphers.

Block ciphers

- Note that stream ciphers operate on the message one bit at a time
- What happens in a stream cipher if you change just one bit of the plaintext?
- We can also use block ciphers
  - Block ciphers operate on the message one block at a time
  - Blocks are usually 64 or 128 bits long
- AES is the block cipher everyone should use today
  - Unless you have a really, really good reason

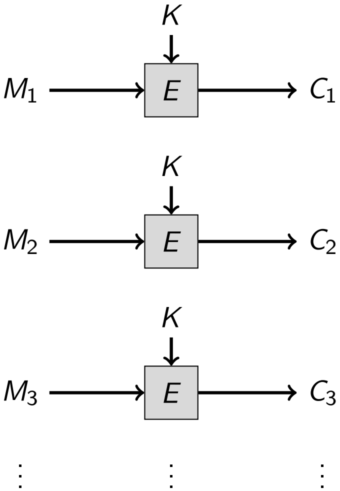
Block ciphers

- Block ciphers work like this:



- If the plaintext is smaller than one block: padding.
- If the plaintext is larger than one block: the choice of what to do with multiple blocks is called the mode of operation of the block cipher

Electronic Code Book (ECB) mode

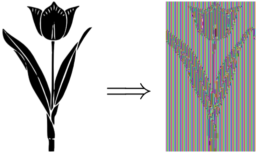


- Electronic Code Book (ECB): encrypt each successive block separately

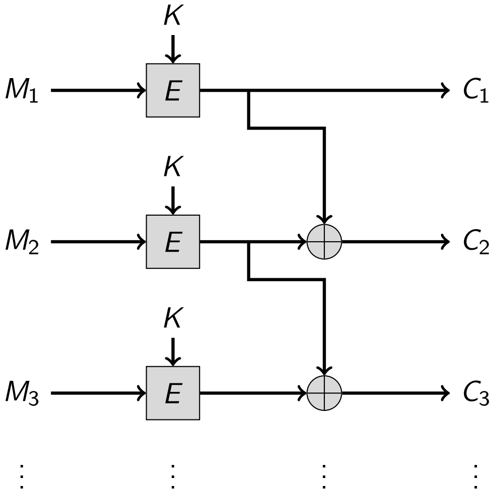
Q: What happens if the plaintext M has some blocks that are identical, $M_i = M_j$?

A: $C_i = E_K(M_i),\ C_j = E_K(M_j) \implies C_i = C_j$

- This reveals patterns in the ciphertext...



Improving ECB (v1)

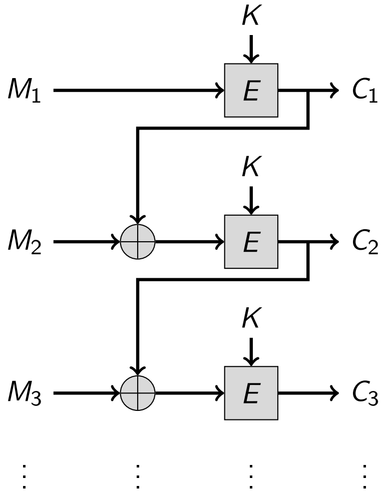


- We can provide “feedback” among different blocks, to avoid repeating patters.

Q: Does this avoid repeating patterns? Any issues here?

A: We can un-do the XOR if we get all the ciphertexts. This basically does not improve compared to ECB.

Improving ECB (v2)



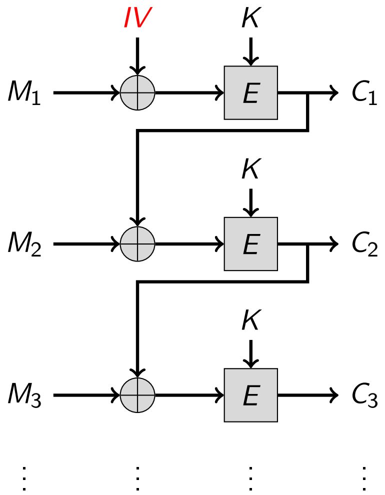
Q: Does this avoid repeating patterns among blocks? Any issues here?

Q: What would happen if we encrypt the message twice with the same key?

A: $C_1 = E_K(M),\ C_2 = E_K(M) \implies C_1 = C_2$

- We could change the key... but there's a better way

Cipher Block Chaining (CBC) mode



Q: Does this solve the issue of encrypting equal blocks? Does this solve the issue of encrypting equal messages/plaintexts?

A: Yes! This is called CBC mode

Q: Can we share IV in the clear?

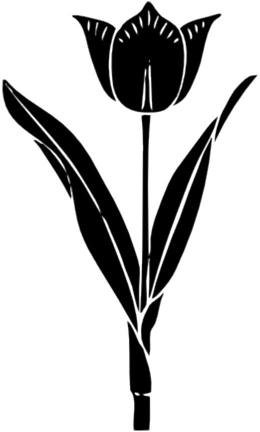
A: Yes!!

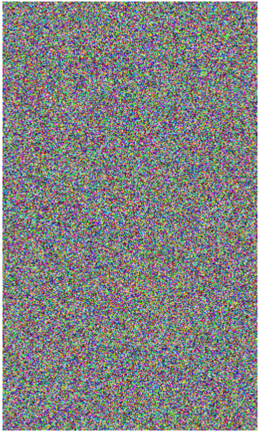
An initialization vector might also be called as a nonce (number used once) or a salt.

Modes of operation

- There are different modes of operation. Common ones include Cipher Block Chaining (CBC), Counter (CTR), and Galois Counter (GCM) modes

- Patterns in the plaintext are no longer exposed because these modes involve some kind of "feedback" among different blocks.
- But you need an IV





Key exchange

- How do Alice and Bob share the secret key?
  - Meet in person; diplomatic courier
  - In general this is very hard
- Or, we invent new technology...

Module outline

1. Introduction to cryptography
2. Secret-key Cryptography
3. Public-key Cryptography
4. Integrity
5. Authentication

Public-key cryptography

- Invented (in public) in the 1970's
- Also called asymmetric cryptography
  - Allows Alice to send a secret message to Bob without any prearranged shared secret!
  - In secret-key cryptography, the same (or a very similar) key encrypts the message and also decrypts it
  - In public-key cryptography, there's one key for encryption, and a different key for decryption!
- Some common examples:
  - RSA, ElGamal, ECC, NTRU, McEliece

Public-key cryptography

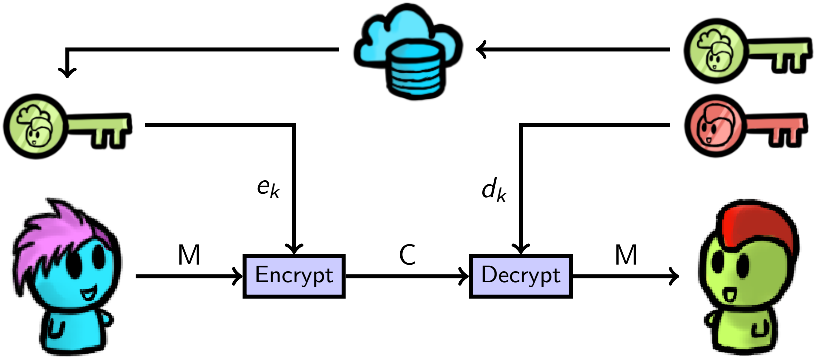
How does it work?

1. Bob creates a key pair (e_k, d_k)
2. Bob gives everyone a copy of his public encryption key e_k
3. Alice uses it to encrypt a message, and sends the encrypted message to Bob
4. Bob uses his private decryption key d_k to decrypt the message
    - Eve can't decrypt it; she only has the encryption key e_k
    - Neither can Alice!
    - It must be hard to derive d_k from e_k

So with this, Alice just needs to know Bob's public key in order to send him secret messages

- These public keys can be published in a directory somewhere

Public-key cryptography



Textbook RSA

- First popular public-key encryption method (published in 1977)
- Relies on the practical difficulty of the factoring problem: given the product of two large prime numbers $n = p \cdot q$, it is very hard to factor n.
- Modular arithmetic: integer numbers that "wrap around"
- High-level idea:
  - It is easy to find large integers e, d, and n, such that:

$$(m^e)^d \equiv m \pmod{n}$$

  - But knowing e and n (and even m), it is extremely hard to find d.

Textbook RSA (simplified)

- Choose two large primes p and q (these are secret).
- Compute $n = p \cdot q$.
- "Choose" a number e, and find d such that

$$(m^e)^d \equiv m \pmod{n}$$

- (This is easy since we know p and q)
- Public key: (e, n)
- Private key: d (other numbers can be discarded)
- Encryption: $c \equiv m^e \pmod{n}$
- Decryption: $c^d \pmod{n}$

- Note that the decryption works.
- Factoring n breaks this!
- This is textbook RSA, never do this!! (we'll see one of the reasons next)

Example of Textbook RSA

- Choose primes p and q
- $n = pq$
- Find e and d (using "math magic")
- Public key: (e, n)
- Private key: d
- Encryption: $c \equiv m^e \pmod{n}$
- Decryption: $m \equiv c^d \pmod{n}$

Q: Example (very small RSA): $p = 53$, $q = 101$, $e = 139$, $d = 1459$.

- Compute n.
- Compute $C_1 = E_e(1011)$. Verify the decryption works.
- Compute $C_2 = E_e(4)$. Verify the decryption works.
- Compute $D_d(C_1 \cdot C_2)$. What is happening? Why?

A: The decryption is the product of the original plaintexts. $(m_1)^e \cdot (m_2)^e \equiv (m_1 \cdot m_2)^e$.

Malleability: it is possible to transform a ciphertext into another ciphertext that decrypts to a related plaintext.
This is typically (but not always!) undesirable.

Chosen Ciphertext Attack

- Choose primes p and q
- $n = pq$
- Find e and d (using "math magic")
- Public key: (e, n)
- Private key: d
- Encryption: $c \equiv m^e \pmod{n}$
- Decryption: $m \equiv c^d \pmod{n}$

Chosen Ciphertext Attack

- We are Eve. Alice is using RSA and her public key is (e, n).
- Bob sends a super-secret message m, encrypted as $c = E_e(m)$. We intercept c.
- Alice is convinced her textbook RSA is very secure, so she is willing to decrypt any ciphertext we send her (except for c), and send us the decryption back.
- Can we ask Alice to decrypt something else that helps us guess m?

Chosen Ciphertext Attack

- Choose primes p and q
- $n = pq$
- Find e and d (using "math magic")
- Public key: (e, n)
- Private key: d
- Encryption: $c \equiv m^e \pmod{n}$
- Decryption: $m \equiv c^d \pmod{n}$

Chosen Ciphertext Attack: solution

- Alice's public key (e, n).
- Bob sends $c_1 = E_e(m)$. We intercept c_1.
- We ask Alice to decrypt, e.g., $c_2 = 2^e \cdot c_1$.

- This decryption yields $(2^e \cdot c_1)^d \equiv 2m$.
- We divide the result by 2, and we get m.

- Textbook RSA is vulnerable against chosen ciphertext attacks (among other things)
- We can fix this with padding techniques (OAEP).

Public key sizes

- Recall that if there are no shortcuts, Eve would have to try 2^{128} things in order to read a message encrypted with a 128-bit key
- Unfortunately, all of the public-key methods we know do have shortcuts
  - Eve could read a message encrypted with a 128-bit RSA key with just 2^{33} work, which is easy!
  - If we want Eve to have to do 2^{128} work, we need to use a much longer public key

Public key sizes

Comparison of key sizes for roughly equal strength

AES	RSA	ECC
80	1024	160
116	2048	232
128	2600	256
160	4500	320
256	14000	512

Hybrid cryptography

- Secret-key cryptography: shorter keys, faster, same key to encrypt and decrypt.
- Public-key cryptography: longer keys, slower, different key to encrypt and decrypt.
- But public-key cryptography is very convenient!
- We can get the best of both worlds:
    - Pick a random 128-bit key K for a secret-key cryptosystem
    - Encrypt the large message with the key K (e.g., using AES)
    - Encrypt the key K using a public-key cryptosystem
    - Send the encrypted message and the encrypted key to Bob
- This hybrid approach is used for almost every cryptography application on the Internet today

Knowledge check!

- Public-key params: (e_A, d_A)
- Secret-key params: K

- Public-key params: (e_B, d_B)
- Secret-key params: ?

- Encrypt/decrypt functions: $E_{key}(\cdot)$, $D_{key}(\cdot)$
- Alice wants to send a large message m to Bob.

Q: How does Alice build the message? How does Bob recover the message?

- Remember: public-key crypto is slow!! We don't want to use it on m.

A: Alice computes $c_1 = E_{e_B}(K)$, $c_2 = E_K(m)$ and sends $\langle c_1 || c_2 \rangle$.
Bob recovers $K = D_{d_B}(c_1)$ and then $m = D_K(c_2)$.

Is that all there is?

- We know how to “send secret messages”, and Eve cannot do anything about it. What else is there to do?
  - Mallory can modify our encrypted messages in transit!
  - Mallory won’t necessarily know what the message says, but can still change it in an undetectable way
    - e.g. bit-flipping attack on stream ciphers
  - This is counterintuitive, and often forgotten
- How do we make sure that Bob gets the same message Alice sent?

Module outline

1. Introduction to cryptography
2. Secret-key Cryptography
3. Public-key Cryptography
4. Integrity
5. Authentication

Integrity components

- How do we tell if a message has changed in transit?
- Simplest answer: use a checksum
  - For example, add up all the bytes of a message
  - The last digits of serial numbers (credit card, ISBN, etc.) are usually checksums
  - Alice computes the checksum of the message, and sticks it at the end before encrypting it to Bob. When Bob receives the message and checksum, he verifies that the checksum is correct.

Checksum does not work!

- With most checksum methods, Mallory can easily change the message in such a way that the checksum stays the same
- We need a “cryptographic” checksum
- It should be hard for Mallory to find a second message with the same checksum as any given one

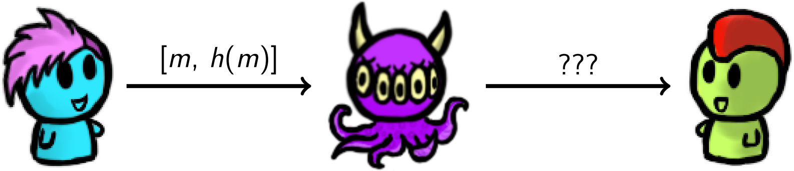
Cryptographic hash functions

- A hash function h takes an arbitrary length string x and computes a fixed length string $y = h(x)$ called a message digest
  - Common examples: MD5, SHA-1, SHA-2, SHA-3 (AKA Keccak, from 2012 on)
- Hash functions should have three properties:
  1. Preimage-resistance:
     - Given y, it's hard to find x such that $h(x) = y$ (i.e., a "preimage" of x)
  2. Second preimage-resistance:
     - Given x, it's hard to find $x' \neq x$ such that $h(x) = h(x')$ (i.e., a "second preimage" of $h(x)$). Note that x is fixed, we have to find x'.
  3. Collision-resistance:
     - It's hard to find any two distinct values x, x' such that $h(x) = h(x')$ (a "collision"). Note that we have free choice of x and x'.

What is “hard”?

- For SHA-1, for example, it takes 2^{160} work to find a preimage or second preimage, and 2^{80} work to find a collision using a brute-force search
  - However, there are faster ways than brute force to find collisions in SHA-1 or MD5
- Collisions are always easier to find than preimages or second preimages due to the well-known birthday paradox
  - If there are n people in a room, what is the probability that at least two people have the same birthday?
  - For 23 people, the probability is larger than 50%!
  - For 40 people, it’s almost 90%!!
  - For 60 people, it’s more than 99%!!!

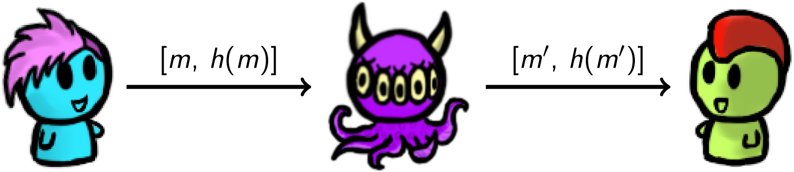
Let’s use a hash function!



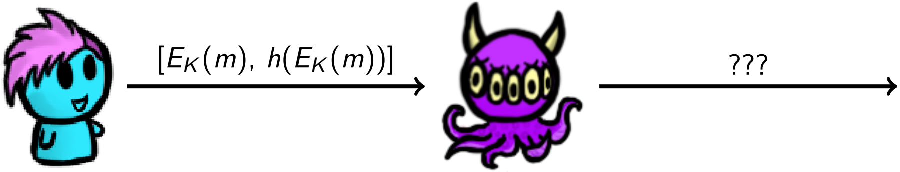
- Assume we don’t care about confidentiality now, just integrity.

Q: What can Mallory do to change the message?

A: Just change it and compute the new message digest herself!



Let’s use a hash function!



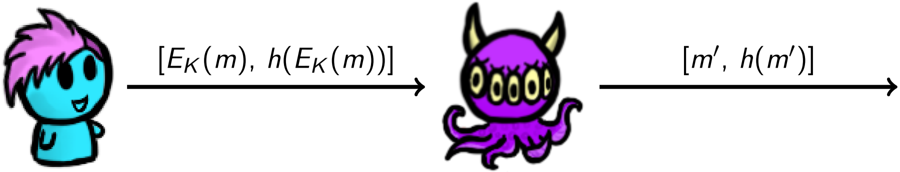
$[E_K(m),\ h(E_K(m))]$

???

- Now we also care about confidentiality

Q: What can Mallory do to change the message?

A: Just change it and compute the new message digest herself!



$[E_K(m),\ h(E_K(m))]$

$[m',\ h(m')]$

Cryptographic hash functions

- Hash functions provide integrity guarantees only when there is a secure way of sending and/or storing the message digest
  - For example, Bob can publish a hash of his public key (i.e., a message digest) on his business card
  - Putting the whole key on there would be too big
  - But Alice can download Bob's key from the Internet, hash it herself, and verify that the result matches the message digest on Bob's card
- What if there's no external channel to be had?
  - For example, you're using the Internet to communicate

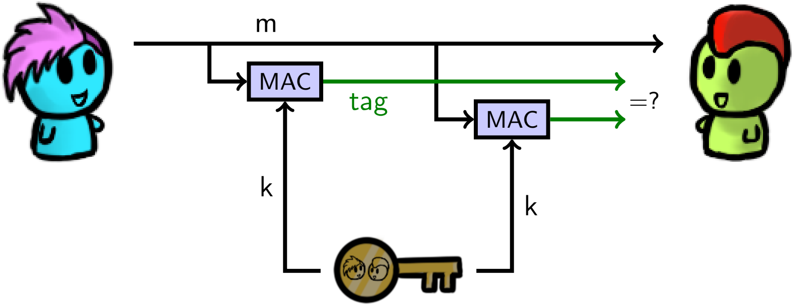
Module outline

1. Introduction to cryptography
2. Secret-key Cryptography
3. Public-key Cryptography
4. Integrity
5. Authentication

Message Authentication Codes (MACs)

- We can use "keyed hash functions", that are usually called Message Authentication Codes, or MACs
- Only those who know the secret key can generate, or even check, the computed hash value (sometimes called a tag)
- Common examples:
  - SHA-1-HMAC, SHA-256-HMAC, CBC-MAC

Message Authentication Codes (MACs)



Combining ciphers and MACs

- In practice we often need both confidentiality and message integrity
- There are multiple strategies to combine a cipher and a MAC when processing a message
    - MAC-then-Encrypt, Encrypt-and-MAC, Encrypt-then-MAC
- Ideally your crypto library already provides an authenticated encryption mode that securely combines the two operations, so you don't have to worry about getting it right
    - E.g., GCM, CCM (used in WPA2, see later), or OCB mode

Combining Ciphers and MACs. Let's try it!

- Alice and Bob have a secret key K for a secret-key cryptosystem $(E_K(\cdot), D_K(\cdot))$ and a secret key K' for their MAC $(MAC_{K'}(\cdot))$. Concatenation is $||$. How does Alice build a message for Bob in the following scenarios?
  - MAC-then-Encrypt: compute the MAC on the message, then encrypt the message and MAC together, and send that ciphertext.

$$E_K(m||MAC_{K'}(m))$$

  - Encrypt-and-MAC: compute the MAC on the message, compute the encryption of the message, and send both.

$$E_K(m)||MAC_{K'}(m)$$

  - Encrypt-then-MAC: encrypt the message, compute the MAC on the encryption, send encrypted message and MAC.

$$E_K(m)||MAC_{K'}(E_K(m))$$

Encrypt and authenticate: what's the right order?

- Usually, we want the receiver to verify the MAC first!

Q: Which of this is the recommended strategy, then?

$$E_K(m||MAC_{K'}(m)), \quad E_K(m)||MAC_{K'}(m), \quad E_K(m)||MAC_{K'}(E_K(m))$$

A: The recommended strategy is Encrypt-then-MAC:

$$E_K(m)||MAC_{K'}(E_K(m))$$

- There is a nice blog post that calls this the "Doom principle": if you have to perform *any* cryptographic operation before verifying the MAC on a message you've received, it will *somehow* inevitably lead to doom.
- It explains two simple attacks that can happen if you violate the Doom principle.

Repudiation

$$\xrightarrow{E_K(m)||MAC_{K'}(E_K(m))}$$

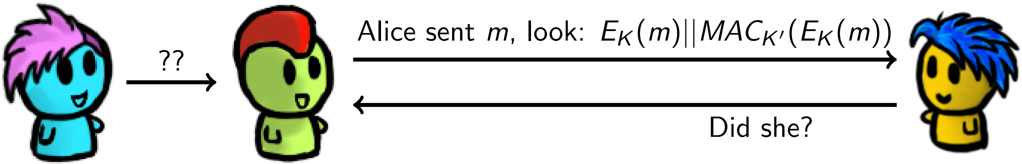
$$\xrightarrow{E_K(m)||MAC_{K'}(E_K(m))}$$

- Bob can be assured that Alice is the one who sent m and that the message has not been modified since she sent it!
- We have confidentiality, integrity, and authentication
- This is like a "signature" on the message... but not quite the same!
- Bob can't prove to Carol that Alice sent m, though.

Q: Why not?

A: Either Alice or Bob could create any of the message and MAC combinations. Also, Carol doesn't know the secret keys.

Repudiation



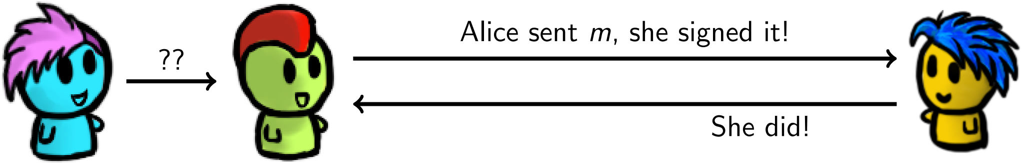
??

Alice sent m, look: $E_K(m)||MAC_{K'}(E_K(m))$

Did she?

- Alice can just claim that Bob made up the message m, and calculated the MAC himself
- This is called repudiation, and we sometimes want to avoid it
- Some interactions should be repudiable
  - Private conversations
- Some interactions should be non-repudiable
  - Electronic commerce

Digital signatures

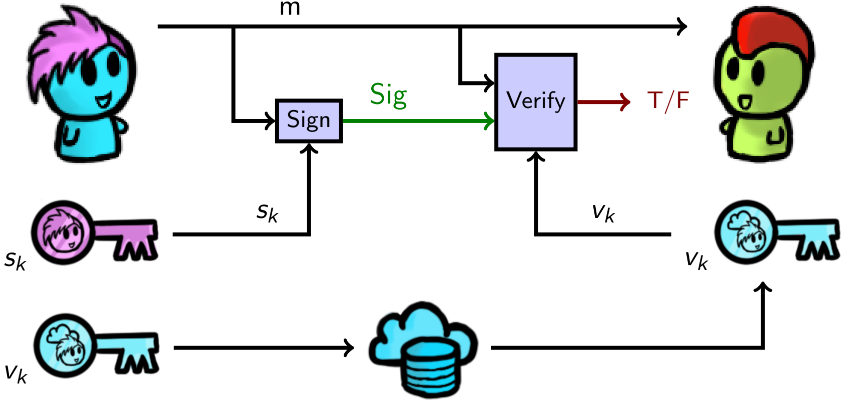


- For non-repudiation, what we want is a true digital signature, with the following properties:
- If Bob receives a message with Alice's digital signature on it, then:
  - Alice, and not an impersonator, sent the message (like a MAC)
  - The message has not been altered since it was sent (like a MAC)
  - Bob can prove these facts to a third party (additional property not satisfied by a MAC).
- How do we arrange this?
  - Use similar techniques to public-key cryptography

Making digital signatures

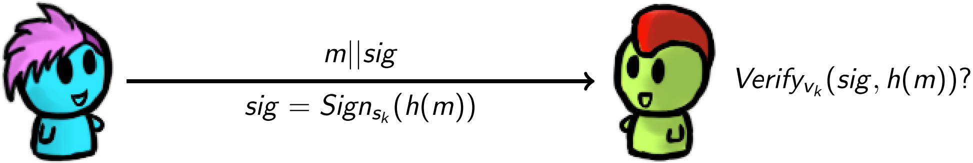
- Remember public-key cryptosystems:
    - Separate keys for encryption and decryption
    - Give everyone a copy of the encryption key
    - The decryption key is private
- To make a digital signature:
    - Alice signs the message with her private signature key (s_k)
- To verify Alice's signature:
    - Bob verifies the message with Alice's public verification key (v_k)
    - If it verifies correctly, the signature is valid

Making digital signatures



Faster signatures

- Just like encryption in public-key crypto, signing large messages is slow
- We can also “hybridize” signatures to make them faster:
  - Alice sends the (unsigned) message, and also a signature on a hash of the message
  - The hash is much smaller than the message, and so it is faster to sign and verify



$m||sig$

$sig = Sign_{s_k}(h(m))$

$Verify_{v_k}(sig, h(m))$?

- Remember that authenticity and confidentiality are separate; if you want both, you need to do both

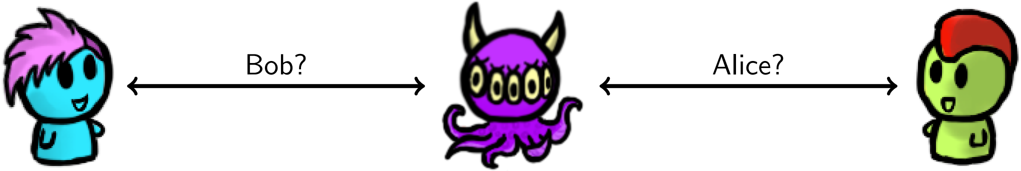
Combining public-key encryption and digital signatures

- Alice has two different key pairs:
  - an (encryption, decryption) key pair (e_k^A, d_k^A)
  - a (signature, verification) key pair (s_k^A, v_k^A)
- So does Bob: (e_k^B, d_k^B) and (s_k^B, v_k^B)
- Alice uses e_k^B to encrypt a message destined for Bob: $C = E_{e_k^B}(M)$
- She uses s_k^A to sign the ciphertext: $T = Sign_{s_k^A}(C)$
- Bob uses v_k^A to check the signature: $Verify_{v_k^A}(C, T)$, if verified, C is authentic
- He uses d_k^B to decrypt the ciphertext: $M = D_{d_k^B}(C)$
- Similarly for reverse direction

Relationship between key pairs

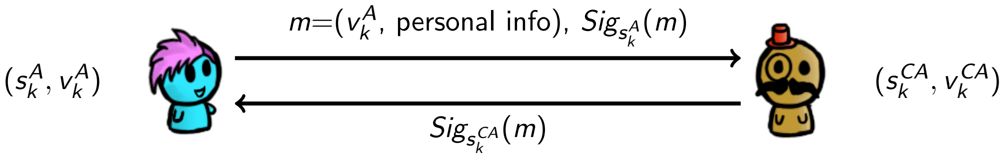
- Alice's (signature, verification) key pair is long-lived, whereas her (encryption, decryption) key pair is short-lived
  - Gives forward secrecy (see later)
- When creating a new (encryption, decryption) key pair, Alice uses her signing key to sign her new encryption key and Bob uses Alice's verification key to verify the signature on this new key

The Key Management Problem



- How can Alice and Bob be sure they're talking to each other, and not Mallory?
- By having each other's verification key.
- Finding this verification key is a very hard problem!
- Possible solutions for Bob to get Alice's verification key:
    - He can know it personally (manual keying)
        - SSH does this
    - He can trust a friend to tell him (web of trust)
        - PGP does this
    - He can trust some third party to tell him (CAs)
        - TLS / SSL do this

Certificate Authorities (CAs)



(s_k^A, v_k^A)

$m{=}(v_k^A$, personal info$)$, $Sig_{s_k^A}(m)$

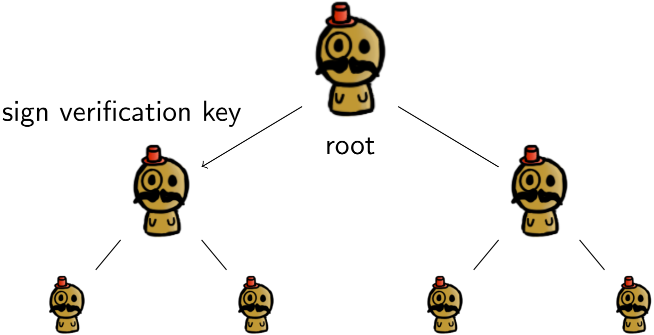
$Sig_{s_k^{CA}}(m)$

(s_k^{CA}, v_k^{CA})

- A CA is a trusted third party who keeps a directory of people's (and organizations') verification keys
- Alice generates a $(s_k^A,\ v_k^A)$ key pair, and sends the verification key and personal information, both signed with Alice's signature key, to the CA
- The CA ensures that the personal information and Alice's signature are correct
- The CA generates a certificate consisting of Alice's personal information, as well as her verification key. The entire certificate is signed with the CA's signature key
- https://letsencrypt.org has changed the game. Most web traffic now encrypted. Extended validation certificates (for which CAs charged a lot of money) now not treated differently by browsers.

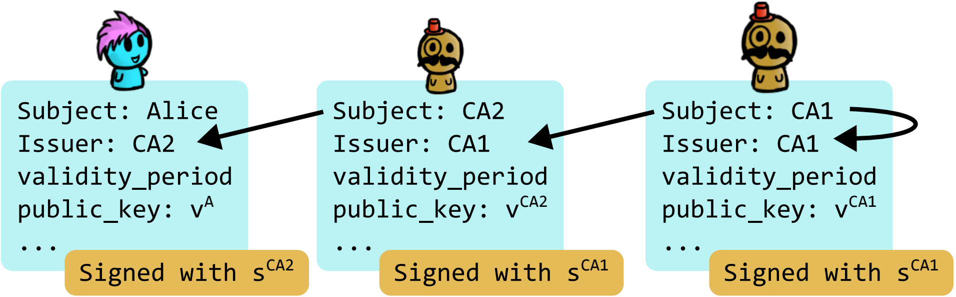
Certificate authorities

- Everyone is assumed to have a copy of the CA's verification key (v_k^{CA}), so they can verify the signature on the certificate
- There can be multiple levels of certificate authorities; level n CA issues certificates for level n+1 CAs – Public-key infrastructure (PKI)
- Need to have only verification key of root CA to verify the certificate chain



Chain of certificates

Alice sends Bob the following certificate to prove her identity. Bob can follow the chain of certificates to validate Alice's identity.



Subject: Alice
Issuer: CA2
validity_period
public_key: v^{A}
...
Signed with s^{CA2}

Subject: CA2
Issuer: CA1
validity_period
public_key: v^{CA2}
...
Signed with s^{CA1}

Subject: CA1
Issuer: CA1
validity_period
public_key: v^{CA1}
...
Signed with s^{CA1}

Bob has v^{CA1}

Putting it all together

- We have all these blocks; now what?
- Put them together into protocols
- This is HARD. Just because your pieces all work, doesn't mean what you build out of them will; you have to *use* the pieces correctly
- Common mistakes include:
  - Using the same stream cipher key for two messages
  - Assuming encryption also provides integrity
  - Falling for replay attacks or reaction attacks
  - *LOTS* more!

Recap: crypto tools

- Secret-key crypto
  - One-time pad
  - Stream ciphers (two-time pad, using nonces)
  - Block ciphers (modes of operation – CBC)
- Public-key crypto
  - Textbook RSA
  - Secret vs. public crypto (speed, key sizes)
  - Hybrid crypto
- Integrity
  - Checksum (usually does not work)
  - Hash functions
- Authentication
  - MACs (repudiation, encrypt-then-MAC)
  - Digital signatures (non-repudiation)
  - Key management
    - Manual keying (SSH)
    - Web of trust (PGP)
    - Certificate authorities (TLS)

CS 458 / 658: Computer Security and Privacy

Module 5 – Security and Privacy of Internet Applications

Part 2 – Cryptography Use Cases (Pt 1)

Fall 2022

Module outline

6 Overview of Security Controls

7 Link-layer Security (WEP, WPA)

8 Network-layer Security (IPSec)

9 Transport-layer Security (TLS)

10 WireGuard

Security controls using cryptography

- We use cryptography as security control in situations where trust cannot be assumed
- We will focus on network security (link layer, network layer, transport layer, and application layer).
- But first, we will see other use cases.

Use cases in program and OS security

- Apps can be installed only if digitally signed by the vendor (BlackBerry) or upgraded only if signed by the original developer (Android)
- OS allows execution of programs only if signed (iOS)
- OS allows loading of certified device drivers only (Windows)
- Secure boot: OS components booted only if correctly signed

Encrypted code

There is research into processors that executes encrypted code only

- The processor will decrypt instructions before executing them
  - Each processor has its own key, we have to encrypt the code for that processor
- The decryption key is processor-dependent
- Malware won't be able to spread without knowing a processor's encryption key

Downsides? It's hard to scale, we have to encrypt the code for a single processor, we don't know the keys of other processors...

Encrypted data

A common technique that aims to protect data in the storage media when the laptop gets lost/stolen, which can be performed either on hardware or by software.

It often does not protect against:

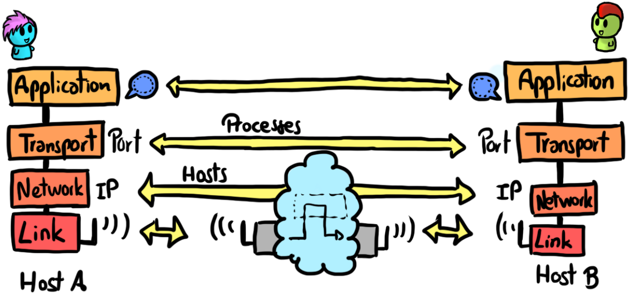
- Other users who legitimately use laptop
- Somebody installing malware on laptop
- Somebody (maybe physically) extracting the decryption key from the laptop's memory

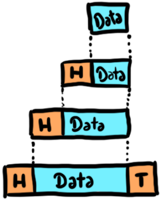
Network security and privacy

Entities you can only communicate with over a network are inherently less trustworthy (e.g., they may not be who they claim to be). This makes networking a primary scenario for cryptography.

This is a separation of concern, and in particular, "*separating the security of the medium from the security of the message*"

Recall the Network Stack





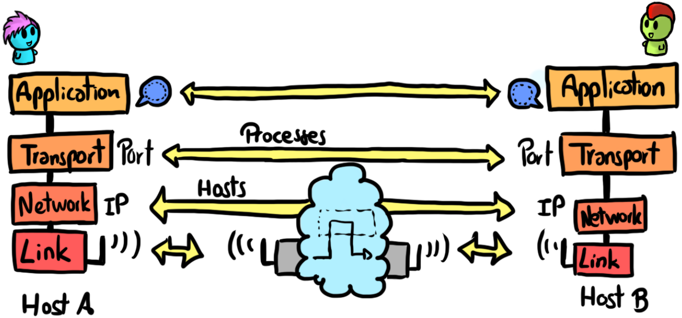
Q: Where do we need to apply crypto? (confidentiality, integrity, authentication)

- A Link layer is enough
- B Application layer is enough
- C We need it in all layers
- D Who needs crypto?

Network security and privacy

Cryptography is used at every layer of the network stack for both security and privacy applications. We will see some examples:

- Link
  - WEP, WPA, WPA2
- Network
  - VPN, IPsec
- Transport
  - TLS/SSL, Tor
- Application
  - ssh, PGP, OTR, Signal, Mixminion



Module outline

6 Overview of Security Controls

7 Link-layer Security (WEP, WPA)

8 Network-layer Security (IPSec)

9 Transport-layer Security (TLS)

10 WireGuard

Wired Equivalent Privacy (WEP)

The Wired Equivalent Privacy (WEP) protocol is a link-layer security protocol that aims to *make wireless communication links just as secure as wired links*.

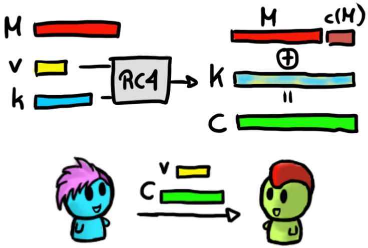
In particular, WEP was intended to enforce three security goals

- Data Confidentiality
  - Prevent an adversary from learning the contents of the wireless traffic
- Data Integrity
  - Prevent an adversary from modifying the wireless traffic or fabricating traffic that looks legitimate
- Access Control
  - Prevent an adversary from using your wireless infrastructure

Unfortunately, none of these is actually enforced!

WEP description

- The sender and receiver share a secret k (either 40 or 104 bits)
- In order to transmit a message M:
  - Compute a checksum $c(M)$ (which does not depend on k)
  - Pick an IV v and generate a keystream $K = RC4(v, k)$
  - Ciphertext $C = K \oplus \langle M \parallel c(M) \rangle$
  - Transmit v and C over the wireless link



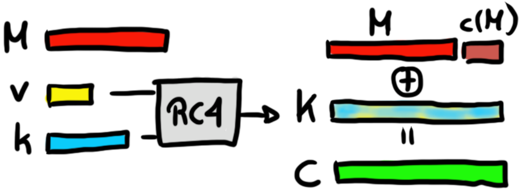
Q: What kind of cipher is this? What does the receiver do with v and C?

A: It's a stream cipher (symmetric)

A: Upon receipt of v and C:

- Use the received v and the shared k for $K = RC4(v, k)$
- Decrypt as $K \oplus C = K \oplus K \oplus \langle M' \parallel c' \rangle = M' \parallel c'$
- Check to see if $c' = c(M')$
- If it is, accept M' as the message transmitted

Problem 1: key reuse



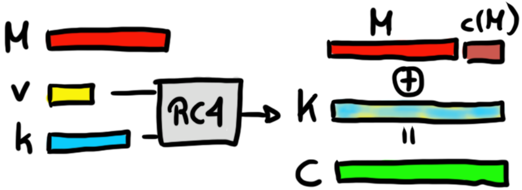
Keystream is derived as: $K = RC4(v, k)$

- IV (v) is too short: only 3 bytes = 24 bits.
- Secret (k) is rarely changed!

Q: What is the problem with this? How could we have avoided this?

A: Key-stream gets re-used after 2^{24} iterations → two-time pad.

Problem 2: integrity breach

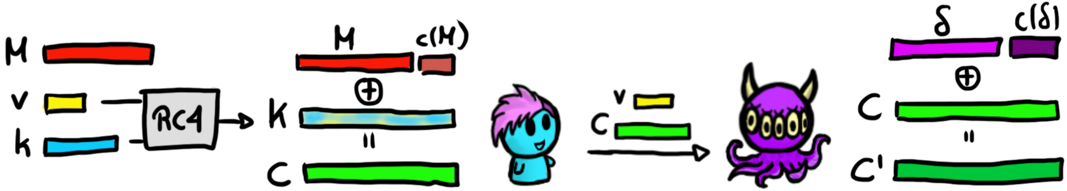


The checksum algorithm in WEP is CRC32, which has two important (and undesirable) properties:

- It is independent of k and v
- It is linear: $c(M \oplus \delta) = c(M) \oplus c(\delta)$

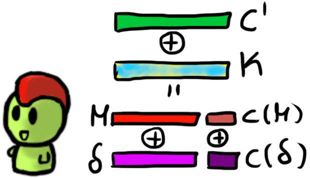
Q: Why is linearity a pessimal property for your integrity mechanism to have when used in conjunction with a stream cipher?

Problem 2: integrity breach



The sender transmits C and v. If Mallory wants to modify the plaintext M into $M' = M \oplus \delta$:

- Calculate $C' = C \oplus \langle \delta \parallel c(\delta) \rangle$
- Send (C', v) instead of (C, v)
- This passes the integrity check of the recipient!

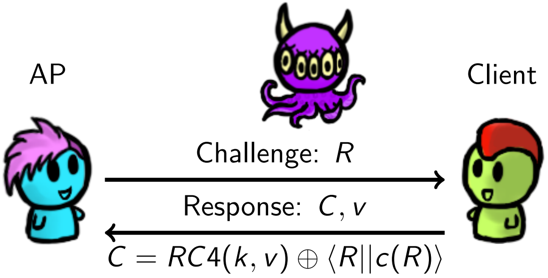


Q: How could we have avoided this?

WEP authentication protocol (disaster)

- WEP's authentication protocol to prove that a client knows k:

- The access point sends a challenge string R to the client
- The client sends back the challenge, WEP-encrypted with the shared secret k
- The wireless access point checks if the challenge is correctly encrypted, and if so, accepts the client



AP

Client

Challenge: R

Response: C, v

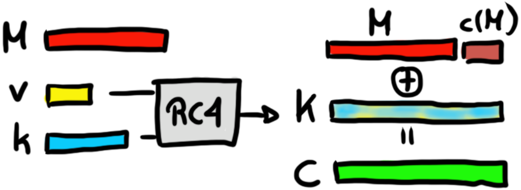
$C = RC4(k, v) \oplus \langle R || c(R) \rangle$

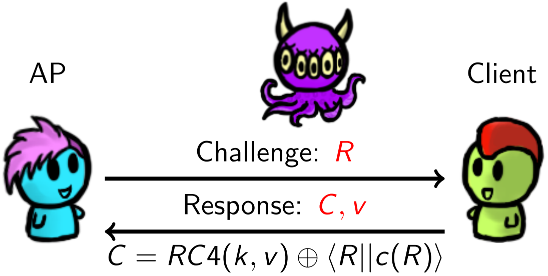
- The adversary has seen both R and (C, v)

Q: What can the adversary do with this?

A: Compute a valid v and $RC4(k, v)$ pair...

Let's think about this...




AP

Client

Challenge: R

Response: C, v

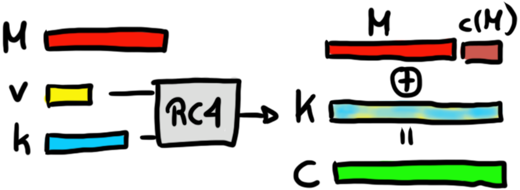
$C = RC4(k, v) \oplus \langle R || c(R) \rangle$

Mallory has seen R, C, and v.

Q: Mallory wants to authenticate herself to the AP. The AP sends Mallory a new challenge R'. Can Mallory successfully run the authentication protocol?

A: Yes! Note that Mallory knows $RC4(k, v) = C \oplus \langle R || c(R) \rangle$. Mallory can just compute: $C' = RC4(k, v) \oplus \langle R' || c(R') \rangle$ and send C' and v to the AP.

Problem 3: packet injection



- Problem 3: Mallory can run the authentication herself (previous slide)
- But, more generally, she can inject packets.
- We saw that seeing R, C, and v gives Mallory a value of v and the corresponding keystream $RC4(v, k)$
- The same way Mallory encrypted the challenge R' in the previous slide, she can encrypt any other value F:
- Then $C' = \langle F \parallel c(F) \rangle \oplus RC4(v, k)$, and she transmits v, C'
- C' is in fact a correct encryption of F, so the message must be accepted

More problems with WEP

- Somewhat surprisingly, the ability to modify and inject packets leads to ways in which Mallory can trick the AP to decrypt packets! Check Prof. Goldberg's talk for more details.
- Note that none of the attacks so far use the fact that the stream cipher was RC4. It turns out that when RC4 is used with similar keys, the output keystream has a subtle weakness, which lead the recovery of either a 104-bit or 40-bit WEP key in under 60 seconds, most of the time. Check this paper for more details.

Replacing WEP

Wi-fi Protected Access (WPA) was rolled out as a short-term patch to WEP while formal standards for a replacement protocol (IEEE 802.11i, later called WPA2) were being developed

- Replaces CRC-32 with a real MAC
- IV is 48 bits
- Key is changed frequently (TKIP)
- Ability to use a 802.1x authentication server
  - But maintains a less-secure PSK (Pre-Shared Key) mode for home users
- Ability to run on most older WEP hardware

Replacing WEP

The 802.11i standard was finalized in 2004, and the result (called WPA2) has been required for products calling themselves "Wi-fi" since 2006

- Replaces the RC4 and MAC algorithms in WPA with the CCM authenticated encryption mode (using AES)
- Considered strong, except in PSK mode
  - Dictionary attacks still possible (avoided in WPA3 (2018))

WEP Recap

Q: What have we learned from WEP?

- Respect to randomness? (provided by IVs?)
- Respect to checksums?

A:

- Use sufficiently long IVs, don't share a key with many people, don't reuse short-term secret keys and IVs.
- Do not use checksums for integrity. Use keyed MACs instead!

You need to understand what was wrong with WEP, how to fix these issues, and you need to be able to identify these issues in other protocols.

Module outline

6 Overview of Security Controls

7 Link-layer Security (WEP, WPA)

8 Network-layer Security (IPSec)

9 Transport-layer Security (TLS)

10 WireGuard

Network layer security: purpose

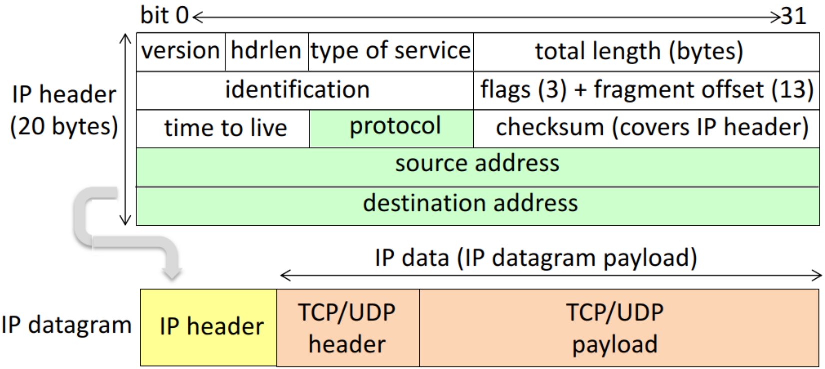
Suppose every link in our network had strong link-layer security. Why would this not be enough?

- Source, destination IPs may not share the same link. Network layer threats such as IP spoofing still exist.
- We need end-to-end security across networks, i.e., securing network layer packets from one host to another so that routers or other hosts in the middle cannot modify or read the packet payload (they still need to read packet metadata for routing)

The IP Security suite (IPSec) extends the Internet Protocol (IP) to provide confidentiality and integrity of packets transmitted across the network. IPSec enables various architectures of Virtual Private Networks (VPNs) which is the foundation in network-layer security.

Recall the IP Datagram

Recall the IP datagram format: no confidentiality, protection against IP spoofing, etc.



bit 0 ↔ 31

IP header (20 bytes)

version | hdrlen | type of service | total length (bytes)

identification | flags (3) + fragment offset (13)

time to live | protocol | checksum (covers IP header)

source address

destination address

IP data (IP datagram payload)

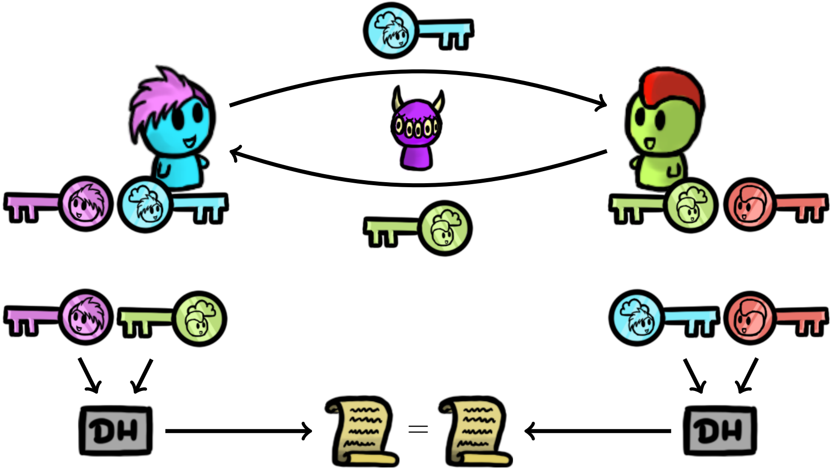
IP datagram: IP header | TCP/UDP header | TCP/UDP payload

IPSec Overview

- Internet Key Exchange (IKE) to agree on a shared symmetric key. We use this key to encrypt and compute MACs over IP packets or parts of it.
- Modes of operation
  - Transport mode
  - Tunnel mode
- Header types
  - Authentication Header (AH)
  - Encapsulated Security Payload (ESP)

Diffie-Hellman Key Exchange Overview

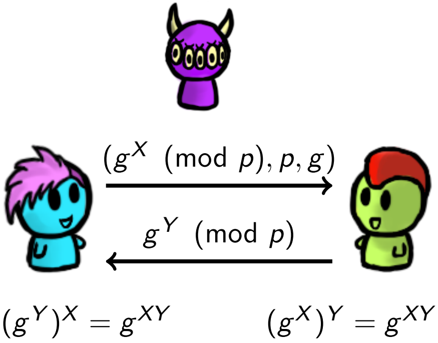
- DH is a public-key protocol that allows two parties to agree on a shared secret over an insecure channel.



Internet Key Exchange (IKE)

The source and destination IP addresses agree on a shared symmetric key via the IKE process, which internally uses the Diffie-Hellman protocol:

- Alice chooses prime p at random and finds a generator g
- Alice chooses $X \leftarrow_R \{2, 3, \ldots, p-2\}$ and sends $A = g^X \pmod{p}$ to Bob, together with p and g
- Bob chooses $Y \leftarrow_R \{2, 3, \ldots, p-2\}$ and sends $B = g^Y \pmod{p}$ to Alice
- Alice and Bob both compute $s = g^{XY} \pmod{p}$
  - Alice does that by computing $B^X \pmod{p}$
  - Bob does that by computing $A^Y \pmod{p}$
- Now they share a common secret s which can be used to derive a symmetric key



$(g^X \pmod{p}), p, g)$

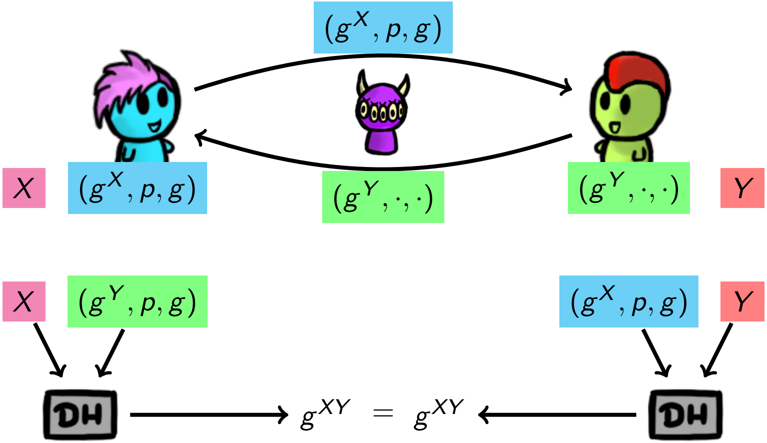
$g^Y \pmod{p}$

$(g^Y)^X = g^{XY}$

$(g^X)^Y = g^{XY}$

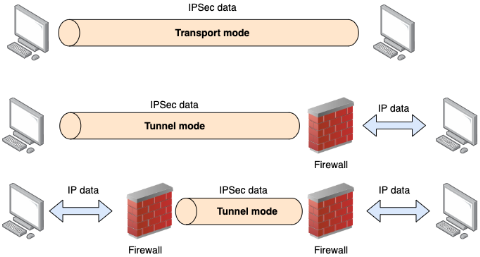
Internet Key Exchange (IKE): more visual

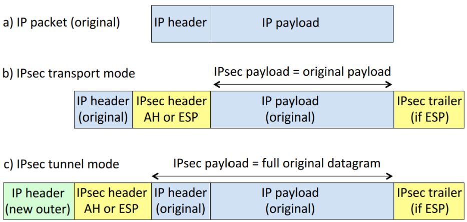
(Background: computing discrete logarithms, e.g., $\log_g Z \pmod{p}$ is very hard!)



Modes of operation

- IPSec has two main modes of operation:
  - Transport mode: uses the original IP header
  - Tunnel mode: encapsulates the original IP header





a) IP packet (original)

IP header | IP payload

b) IPsec transport mode

IPsec payload = original payload

IP header (original) | IPsec header AH or ESP | IP payload (original) | IPsec trailer (if ESP)

c) IPsec tunnel mode

IPsec payload = full original datagram

IP header (new outer) | IPsec header AH or ESP | IP header (original) | IP payload (original) | IPsec trailer (if ESP)

Figure 10.13: IPsec transport mode vs. tunnel mode (structural views).

IPSec Headers

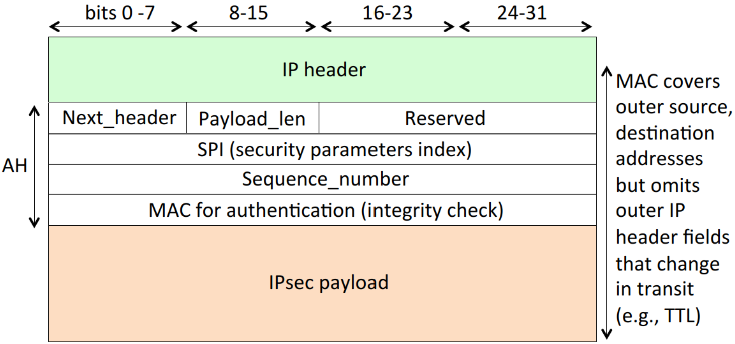
Authentication Header (AH) – RFC4302

- Offers integrity and data source authentication
  - Authenticates payload and parts of IP header that do not get modified during transfer, e.g., source IP address
- Offers protection against replay attacks
  - Via extended sequence numbers

Encapsulated Security Payload (ESP) – RFC4303

- Offers confidentiality
  - IP data is encrypted during transmission
- Offers authentication functionality similar to AH
  - But authenticity checks only focus on the IP payload
- Applies padding and generates dummy traffic
  - Makes traffic analysis harder

IPSec: Authentication Header (AH) view



bits 0 -7

8-15

16-23

24-31

IP header

Next_header | Payload_len | Reserved

SPI (security parameters index)

Sequence_number

MAC for authentication (integrity check)

IPsec payload

AH

MAC covers outer source, destination addresses but omits outer IP header fields that change in transit (e.g., TTL)

Figure 10.11: IPsec Authentication Header (AH) field view, for both transport and tunnel modes. Next_header identifies the protocol of the AH payload (e.g., TCP=6). Payload_len is used to calculate the length of the AH header. SPI identifies the Security Association. Sequence_number allows replay protection (if enabled).

IPSec: Encapsulated Security Payload (ESP) view

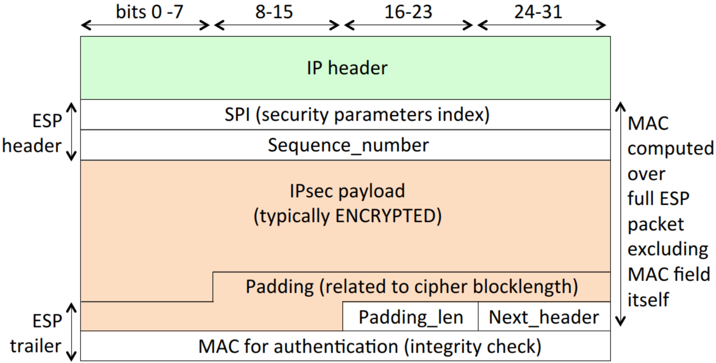

bits 0 -7
8-15
16-23
24-31
IP header
ESP header
SPI (security parameters index)
Sequence_number
IPsec payload
(typically ENCRYPTED)
Padding (related to cipher blocklength)
Padding_len
Next_header
ESP trailer
MAC for authentication (integrity check)
MAC computed over full ESP packet excluding MAC field itself

Figure 10.12: IPsec Encapsulating Security Payload (ESP) field view, for both transport and tunnel modes. `SPI` identifies the Security Association. `Sequence_number` allows replay protection (if enabled). `Next_header` (which may include a crypto IV or Initialization Vector) indicates the type of data in the ENCRYPTED field. A payload length field is not needed, as the ESP header is fixed at two 32-bit words, and the length of the IPsec payload (which is the same as that of the original payload) is specified in the IP header.

IPSec packets' format

A regular IP packet in the form of $\langle\, H \parallel P \,\rangle$ can be transformed into an IPSec packet depending on the mode of operation:

	AH	**ESP**
Transport	$H \parallel AH \parallel P$ $\hookrightarrow$ Int. of H and P	$H \parallel ESP\text{-}H \parallel \langle\, P \,\rangle_k \parallel ESP\text{-}T$ $\hookrightarrow$ Int. and Conf. of P only
Tunnel	$H' \parallel AH \parallel \langle\, H \parallel P \,\rangle$ $\hookrightarrow$ Int. of H and P	$H' \parallel ESP\text{-}H \parallel \langle\, H \parallel P \,\rangle_k \parallel ESP\text{-}T$ $\hookrightarrow$ Int. and Conf. of H and P

The Tunnel-ESP combination (also known as an IP-in-IP tunneling) is often used to implement Virtual Private Networks (VPNs)

IPSec deployment challenges

- Needs to be included in the kernel's network stack.
- There may be legitimate reasons to modify some IP header fields; IPSec breaks networking functionalities that require such changes.
    - with AH, you cannot replace a private address for a public one at a NAT box.
    - with ESP, it depends
        - In transport usually does not work due to TCP and UDP checksums
        - In tunnel mode it is fine
- IPSec is complex, hard to audit, and prone to misconfigurations

Module outline

6 Overview of Security Controls

7 Link-layer Security (WEP, WPA)

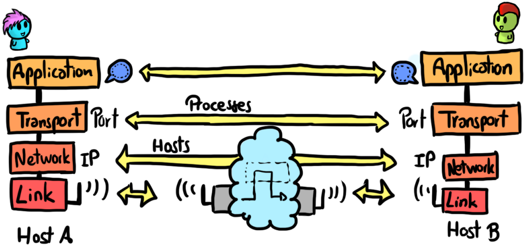
8 Network-layer Security (IPSec)

9 Transport-layer Security (TLS)

10 WireGuard

Transport-layer security and privacy

- Network-layer security mechanisms arrange to send *individual* IP packets securely from one network to another
- Transport-layer security mechanisms transform arbitrary TCP connections to add security and privacy
- The main transport-layer *security* mechanism:
  - TLS (formerly known as SSL)
- The main transport-layer *privacy* mechanism:
  - Tor — will be covered in the lecture on PETs



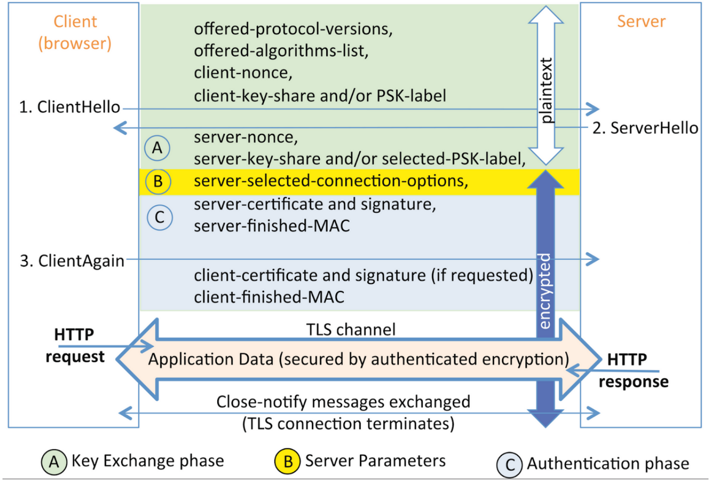
TLS / SSL

- In the mid-1990s, Netscape invented a protocol called Secure Sockets Layer (SSL) meant for protecting HTTP (web) connections
  - The protocol, however, was general, and could be used to protect any TCP-based connection
  - HTTP + SSL = HTTPS
- Historical note: there was a competing protocol called S-HTTP. But Netscape and Microsoft both chose HTTPS, so that's the protocol everyone else followed
- SSL went through a few revisions, and was eventually standardized into the protocol known as TLS (Transport Layer Security, imaginatively enough)

TLS at a high level: RFC8446

- Client connects to server, indicates it wants to speak TLS, with
  - Client key-share under ECDHE
  - The list of ciphersuites it knows
- Server sends its certificate to client, which contains:
  - Server key-share under ECDHE
  - Its host name
  - Its verification key
  - Some other administrative information
  - A signature from a Certificate Authority (CA)
- Both client and server derives the same session key K (which is hard for Eve to derive) based on the two key shares
- Server also chooses which ciphersuite to use
- All remaining traffic will be encrypted and authenticated under K

TLS connection establishment


Client (browser)
Server
offered-protocol-versions,
offered-algorithms-list,
client-nonce,
client-key-share and/or PSK-label
1. ClientHello
plaintext
2. ServerHello
A
server-nonce,
server-key-share and/or selected-PSK-label,
B
server-selected-connection-options,
C
server-certificate and signature,
server-finished-MAC
3. ClientAgain
client-certificate and signature (if requested)
client-finished-MAC
encrypted
HTTP request
TLS channel
Application Data (secured by authenticated encryption)
HTTP response
Close-notify messages exchanged
(TLS connection terminates)
A Key Exchange phase
B Server Parameters
C Authentication phase

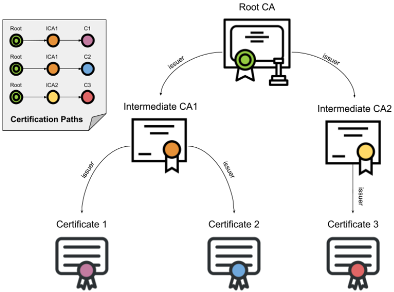
Security properties provided by TLS

- Server authentication
- Message integrity
- Message confidentiality
- Client authentication (optional)

Certification Authorities (CAs) in TLS

A certification authority acts as a trusted third-party that:

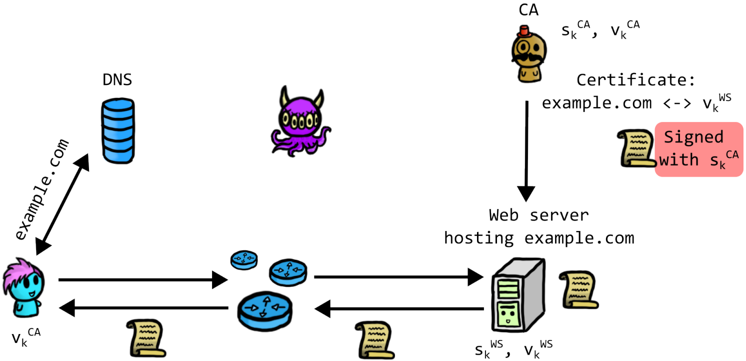
- Issues digital certificates
- Certifies the ownership of a public key by the named subject of the certificate
- Manages certificate revocation lists (CRLs)



What can go wrong with TLS?

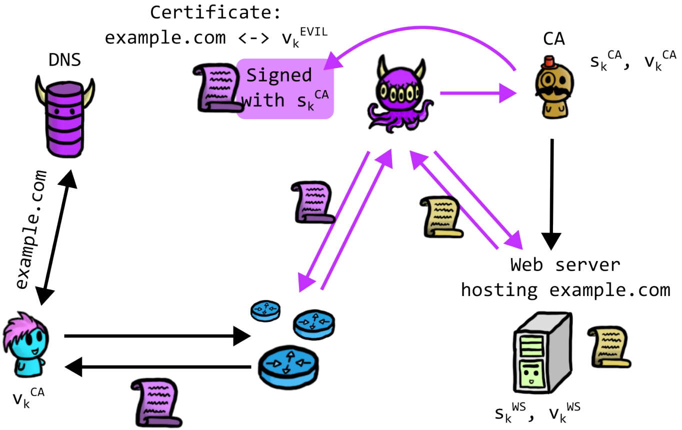
Basic idea: Alice accepts the connection if she receives a certificate and

1. the certificate is signed by a CA she trusts (v_k^{CA})
2. the certificate is for the domain she's requesting
3. when talking to the web server, Alice can verify the signatures with v_k^{WS} (which is in the certificate).



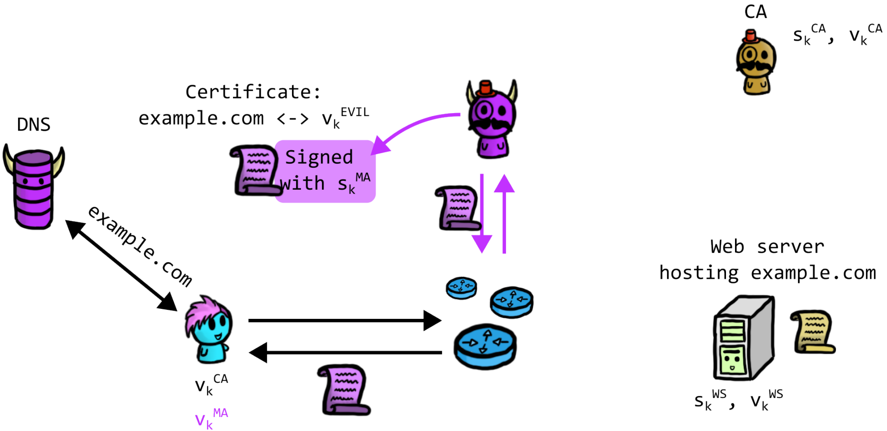
What can go wrong with TLS?

An adversary can compromise a CA to plant fake certificates (e.g., DigiNotar’s fake *.google.com certificates used by an ISP in Iran)



What can go wrong with TLS?

An adversary can install a custom CA on users' devices, allowing them to sign certificates that clients will accept for any site (e.g., in 2019, Kazakhstan's ISPs mandated the installation of a root certificate issued by the government)

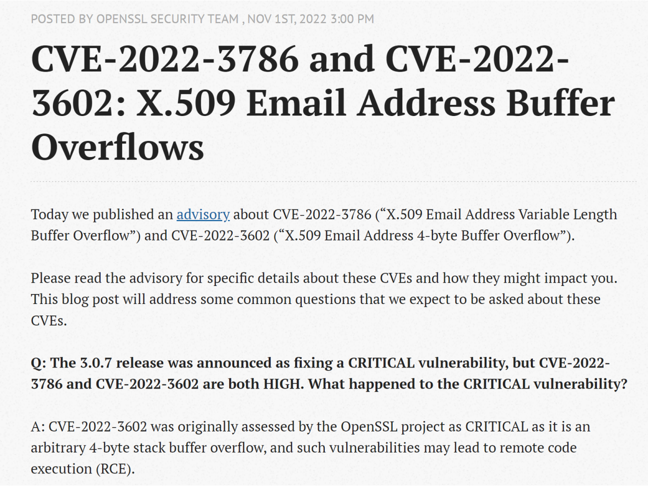


What can go wrong with TLS?

Other examples:

- Companies may think it is an excellent idea
  - e.g., Lenovo's Superfish or Sennheiser HeadSetup root certificates (for advertisement and communication purposes, respectively)
- There have been many issues with TLS/SSL implementations
- Here's a very interesting talk about some of these issues.

This happened just yesterday!



POSTED BY OPENSSL SECURITY TEAM , NOV 1ST, 2022 3:00 PM

CVE-2022-3786 and CVE-2022-3602: X.509 Email Address Buffer Overflows

Today we published an advisory about CVE-2022-3786 (“X.509 Email Address Variable Length Buffer Overflow”) and CVE-2022-3602 (“X.509 Email Address 4-byte Buffer Overflow”).

Please read the advisory for specific details about these CVEs and how they might impact you. This blog post will address some common questions that we expect to be asked about these CVEs.

Q: The 3.0.7 release was announced as fixing a CRITICAL vulnerability, but CVE-2022-3786 and CVE-2022-3602 are both HIGH. What happened to the CRITICAL vulnerability?

A: CVE-2022-3602 was originally assessed by the OpenSSL project as CRITICAL as it is an arbitrary 4-byte stack buffer overflow, and such vulnerabilities may lead to remote code execution (RCE).

SSL-based VPNs

- We can use SSL/TLS to create secure site-to-site tunnels
  - Similarly to IPSec
- A more flexible "user-space VPN"
  - In contrast to IPSec, it does not require kernel-level access
  - Virtual network interfaces are used instead
- Several solutions available:
  - e.g., OpenVPN, Cisco AnyConnect

Module outline

6 Overview of Security Controls

7 Link-layer Security (WEP, WPA)

8 Network-layer Security (IPSec)

9 Transport-layer Security (TLS)

10 WireGuard

Issues with existing VPNs

IPSec:

- Is complex, hard to audit, and prone to misconfigurations
  - Big book of IPSec RFCs: Internet security architecture (Loshin, '99)
- Does not prevent you from making bad choices
  - Supports all ciphers, including obsolete ones and NULL

SSL VPNs:

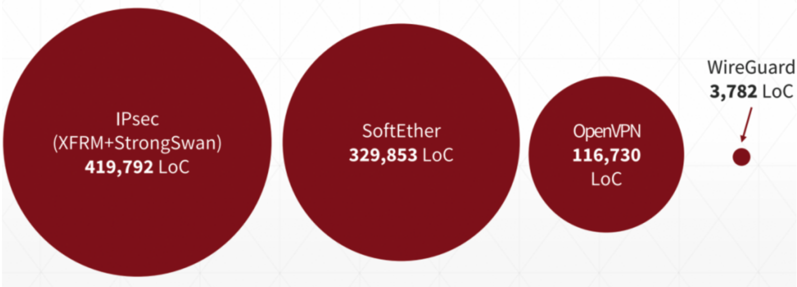
- Also on the complex side
- Tends to be slow
- Also does not prevent you from making bad choices

WireGuard

- New (and simpler) VPN design built from the ground-up
- Offers a kernel and a user-space implementation
- Faster than IPSec and TLS-based VPN solutions

Lightweight and secure

- Easy to configure
    - But no PKI, keys are distributed manually
- Easy to audit
    - 4,000 LoCs vs IPSec's 400,000 LoCs
- Hard to get it wrong
    - Single cipher suite



Recap: cryptography use cases (Pt. I)

- Security controls:
  - Digital signature: app installation, OS execution, drivers, secure boot.
  - Encryption: code encryption, disk encryption.
- Link layer: WEP problems
  - Short IV → two-time pad → make it bigger!
  - Checksum → integrity breach → use MACs
  - Protocol disaster → packet injection
- Network layer: IPSec
  - IKE: Diffie-Hellman
  - Modes: transport, tunnel
  - Headers: AH, ESP
  - Too many parameters
- Transport layer: TLS
  - Protocol summary (ECDHE, etc.)
  - Key management: CAs
  - Issues with TLS: MITM
- Wireguard
  - Better VPN

CS 458 / 658: Computer Security and Privacy

Module 5 – Security and Privacy of Internet Applications

Part 3 – Cryptography Use Cases (Pt 2)

Fall 2022

Module outline

11 Application layer security

12 SSH

13 PGP

14 OTR

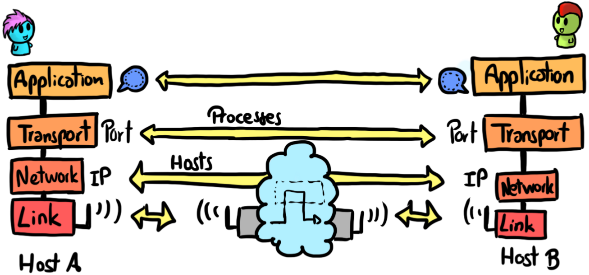
15 Signal

Application layer security

- TLS can provide for encryption at the TCP socket level
  - “End-to-end” in the sense of a network connection
  - Is this good enough? Hint: one application may involve multiple TCP connections
- Many applications would like true end-to-end security
  - Human-to-human would be best, but those last 50 cm are really hard!
  - We usually content ourselves with desktop-to-desktop

We’ll look at three particular applications:

- SSH
- PGP
- Instant messaging (OTR, Signal)



Module outline

11 Application layer security

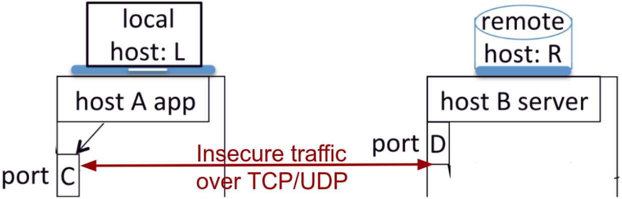
12 SSH

13 PGP

14 OTR

15 Signal

Insecure application traffic pre-SSH

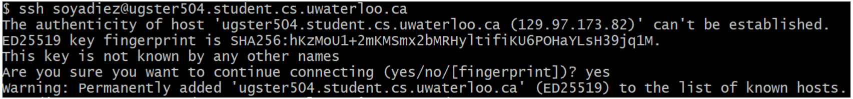


- Suppose that you want to connect to a remote machine
  - You may think “Oh ok, let me use Telnet”
- Think again...
  - All data exchanged through Telnet is in plain text!

Enter secure remote login (ssh)

- You're already familiar with this tool for securely logging in to a remote computer (the ugster machines)
- Usual usage (simplified):
  - Client connects to server
  - Server sends its verification key
    - The client should verify that this is the correct key

  Example: I was trying to connect to an `ugster` machine for the first time:



```
$ ssh soyadiez@ugster504.student.cs.uwaterloo.ca
The authenticity of host 'ugster504.student.cs.uwaterloo.ca (129.97.173.82)' can't be established.
ED25519 key fingerprint is SHA256:hKzMoU1+2mKMSmx2bMRHyltifiKU6POHaYLsH39jq1M.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'ugster504.student.cs.uwaterloo.ca' (ED25519) to the list of known hosts.
```

Q: Have you ever verified this fingerprint?

  - Many clients implement Trust on first use (TOFU):
    - The client software prompts to confirm the connection.
    - Subsequent connections are fine unless the public key of the server has changed.

Enter secure remote login (ssh)

- Usual usage (simplified):
  - Client connects to server
  - Server sends its verification key
    - The client should verify that this is the correct key
  - Client and server run a key agreement protocol to establish session keys, server signs its messages
    - All communication from here on in is encrypted and MAC-ed with the session keys
  - *Client authenticates to server*
  - Server accepts authentication, login proceeds

Client authentication with ssh

There are two main ways to authenticate with ssh:

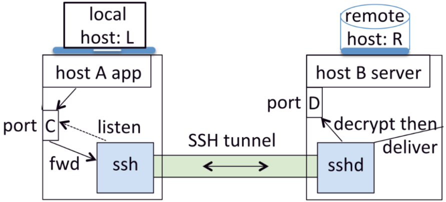
- Send a password over the encrypted channel
  - The server needs to know (a hash of) your password
- Sign a random challenge with your private signature key
  - The server needs to know your public verification key

Q: Advantages/disadvantages of each?

A: People create weak passwords, people write their passwords in post-it notes, etc.

A: People usually don't protect private keys with passphrases

SSH port forwarding



SSH allows for tunneling:

- The client machine can create a mapping between a local TCP port and a port in the remote machine
  - e.g., localhost:IMAP to mail.myorg.ca:IMAP
- The client SSH and the server SSHd operate as a secure relay
  - Allows the client to interact with server applications via SSH

Module outline

11 Application layer security

12 SSH

13 PGP

14 OTR

15 Signal

Pretty Good Privacy

- The first popular implementation of public-key cryptography.
- Originally made by Phil Zimmermann in 1991
  - He got in a lot of trouble for it, since cryptography was highly controlled at the time.
  - But that's a whole 'nother story. :-)
- Today, there are many (more-or-less) compatible programs
  - GNU Privacy Guard (gpg), Hushmail, etc.

Pretty Good Privacy

- What does it do?
  - Its primary use is to protect the contents of email messages.
  - Provides confidentiality, integrity, authentication, and non-repudiation.

Q: What do we use for non-repudiation?

A: Digital signatures!

- How does it work?
  - Uses public-key cryptography to provide:
    - Encryption of email messages (using hybrid encryption)
    - Digital signatures on email messages (hash-then-sign)

Recall: Public-key Cryptography, Sign-then-Encrypt

- encryption/decryption: (e_A, d_A)
- signature/verification: (s_A, v_A)

- encryption/decryption: (e_B, d_B)
- signature/verification: (s_B, v_B)

To send a message to Bob, Alice will:

- Write a message
- Sign it with ???
- Encrypt both the message and the signature with ???, send them to Bob

Q: You know how to write that at this point, right? (what are the ???)

A: $E_{e_B}(m \| Sign_{s_A}(m))$

Q: And you also know what Bob does next...

But you said we prefer to “authenticate last”!

- Both Encrypt-then-sign and Sign-then-encrypt have their uses
- We saw “authenticating last” can prevent certain attacks but...

Q: What can Eve learn from an Encrypt-then-Sign message that she cannot learn from a Sign-then-Encrypt message?

$$E_{e^B}(m) \parallel Sign_{s^A}(E_{e^B}(m)) \qquad E_{e^B}(m \parallel Sign_{s^A}(m))$$

A: Eve can see Alice signed the encrypted message (if she has Alice’s verification key)

But you said we prefer to "authenticate last"!

- Both Encrypt-then-sign and Sign-then-encrypt have their uses
- We saw "authenticating last" can prevent certain attacks but...

Q: What can Mallory do with a captured Encrypt-then-Sign message?

$$E_{e_B}(m) \parallel Sign_{s_A}(E_{e_B}(m)) \qquad E_{e_B}(m \parallel Sign_{s_A}(m))$$

A: Mallory could remove the signature and sign it herself! (even if she does not know the plaintext)

$$E_{e_B}(m) \parallel Sign_{s_A}(E_{e_B}(m)) \rightarrow E_{e_B}(m) \parallel Sign_{s_M}(E_{e_B}(m))$$

So... what do we do?

- The “modern approach” to confidentiality, integrity, and authentication uses authenticated encryption (and more advanced tools that we are not seeing in this course).
- What does PGP do?
  - **PGP**: original protocol from 1991
  - **OpenPGP**: open-source version, from 1997, updated
  - **GPG** (GNU Privacy Guard): most common implementation, follows OpenPGP
- People usually refer to OpenPGP when talking about “PGP”
- OpenPGP offers now modern solutions for authenticated encryption, but it is also very complex (similar to what we have seen with IPSec and TLS...)
- You do not need to know all this; we are going to focus on how PGP deals with the *key management problem*

Back to PGP

PGP's main functions:

- Create these four kinds of keys
  - encryption, decryption, signature, verification
- Encrypt messages using someone else's encryption key
- Decrypt messages using your own decryption key
- Sign messages using your own signature key
- Verify signatures using someone else's verification key
- Sign other people's keys using your own signature key
- Disclaimer: in practice there are primary keypairs, used for signing, verifying, and creating encryption sub-keypairs, but let's abstract from this...

Obtaining keys

Earlier, we said that Alice needs to get an authentic copy of Bob's public key in order to send him an encrypted message.

How does Alice do this?

Q: We could use Certificate Authorities (CAs), like in...

A: TLS / SSL

Q: ... or manual keying like in...

A: SSH

What if we don't involve CAs?

- Bob could put a copy of his public key on his webpage
- Is this good enough?

Verifying public keys

- The key in Bob's website would look something like this:

```
mQGiBDi5qEURBADitpDzvvzW+9lj/zYgK78G3D76hvvvIT6gpTIlwg6WIJNLKJat
O1yNpMIYNvpwi7EUd/lSNl6t1/AO22p7s7bDbE4T5NJdaOIOAgWeOZ/plIJC4+o2
tD2RNuSkwDQcxzm8KUNZOJla4LvgRkm/oUubxyeY5omus7hcfNrBOwjC1wCg4Jnt
m7s3eNfMu72Cv+6FzBgFog8EANirkNdC1Q8oSMDihWj1ogiWbBz4s6HMxzAaqNf/
rCJ9qoK5SLFeoB/r5ksRWty9QKV4VdhhCIy1U2B9tSTlEPYXJHQPZ3mwCxUnJpGD
8UgFM5uKXaEq2pwpArTm367k0tTpMQgXAN2HwiZv//ahQXH4ov30kBBVL5VFxMUL

UJ+yA/4r5HLTpP2SbbqtPWdeW7uDwhe2dTqffAGuf0kuCpHwCTAHr83ivXzT/7OM
```

- If Alice knows Bob personally, she could:
  - Download the key from Bob's web page
  - Phone up Bob, and verify she's got the right key
  - Problem: keys are big and unwieldy!

Fingerprints

- Luckily, there's a better way!
- A fingerprint is a cryptographic hash of a key
- This, of course, is *much shorter*:
  - B117 2656 DFF9 83C3 042B C699 EB5A 896A 2898 8BF5

Q: Can Eve generate another key-pair whose public key has the same fingerprint?

A: Collision-resistant hash functions: there's no (known) way to make two different keys that have the same fingerprint

Fingerprints

- So now we can try this:
  - Alice downloads Bob's key from his webpage
  - Alice's software calculates the fingerprint
  - Alice phones up Bob, and asks him to read his key's actual fingerprint to her
  - If they match, Alice knows she's got an authentic copy of Bob's key
- That's great for Alice, but what about Carol?
  - Carol might not know Bob
  - At least not well enough to phone him

Signing keys

- Once Alice has verified Bob's key, she uses her signature key to sign Bob's key, and link it to Bob's email address
- This is effectively the same as Alice signing a message that says:

  "I have verified that the key with fingerprint
  `B117 2656 DFF9 83C3 042B C699 EB5A 896A 2898 8BF5`
  really belongs to Bob (bob@bobmail.com)"
- Bob can attach Alice's signature to the key on his webpage
  - If Bob wants, he can get many people to sign his key...

Q: Can you see some potential issue with key signing?

A: Once you sign a key... you cannot take that back...

Web of Trust

- Now Alice can act as an introducer for Bob
- If Carol doesn't know Bob, but does know Alice (and has already verified Alice's key, and trusts her to introduce other people):
  - she downloads Bob's key from his website
  - she sees Alice's signature on it
  - she is able to use Bob's key without having to check with Bob personally
- This is called the Web of Trust, and the PGP software handles it mostly automatically

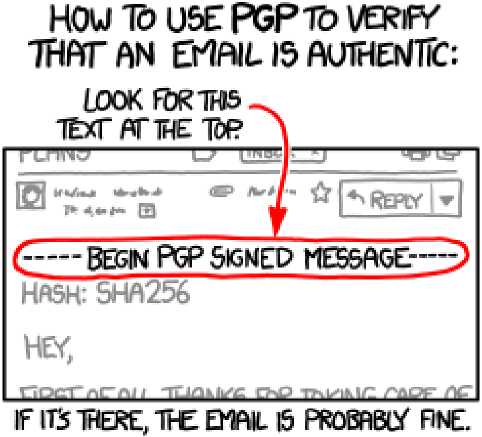
So, great!

So if Alice and Bob want to have a private conversation by email:

- They each create their sets of keys
- They exchange public encryption keys and verification keys
- They send signed and encrypted messages back and forth
- (You can also use encryption or signing independently)

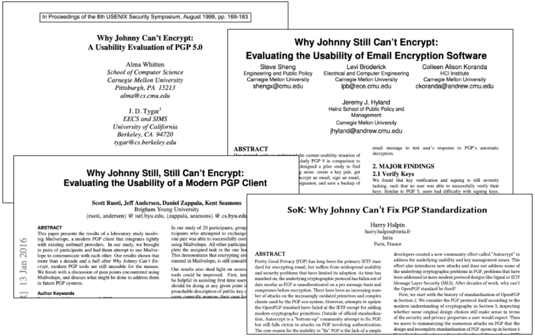
Pretty Good, no?

How to use PGP


HOW TO USE PGP TO VERIFY THAT AN EMAIL IS AUTHENTIC:
LOOK FOR THIS TEXT AT THE TOP:
REPLY
-----BEGIN PGP SIGNED MESSAGE-----
HASH: SHA256
HEY,
FIRST OF ALL, THANKS FOR TAKING CARE OF
IF IT'S THERE, THE EMAIL IS PROBABLY FINE.

(If you want to be extra safe, check that there's a big block of jumbled characters at the bottom.)

Problem 1: Usability



In Proceedings of the 8th USENIX Security Symposium, August 1999, pp. 169-183

Why Johnny Can't Encrypt:
A Usability Evaluation of PGP 5.0

Alma Whitten
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
alma@cs.cmu.edu

J. D. Tygar[1]
EECS and SIMS
University of California
Berkeley, CA 94720
tygar@cs.berkeley.edu

Why Johnny Still Can't Encrypt:
Evaluating the Usability of Email Encryption Software

Steve Sheng, Engineering and Public Policy, Carnegie Mellon University, shengx@cmu.edu
Levi Broderick, Electrical and Computer Engineering, Carnegie Mellon University, lpb@ece.cmu.edu
Colleen Alison Koranda, HCI Institute, Carnegie Mellon University, ckoranda@andrew.cmu.edu
Jeremy J. Hyland, Heinz School of Public Policy and Management, Carnegie Mellon University, jhyland@andrew.cmu.edu

ABSTRACT

email message to test user's response to PGP's automatic decryption.

2. MAJOR FINDINGS

2.1 Verify Keys

We found that key verification and signing is still severely lacking, such that no user was able to successfully verify their keys. Similar to PGP 5, users had difficulty with signing keys.

Why Johnny Still, Still Can't Encrypt:
Evaluating the Usability of a Modern PGP Client

Scott Ruoti, Jeff Andersen, Daniel Zappala, Kent Seamons
Brigham Young University
{ruoti, andersen} @ isrl.byu.edu, {zappala, seamons} @ cs.byu.edu

13 Jan 2016

ABSTRACT

This paper presents the results of a laboratory study involving Mailvelope, a modern PGP client that integrates tightly with existing webmail providers. In our study, we brought in pairs of participants and had them attempt to use Mailvelope to communicate with each other. Our results shown that more than a decade and a half after *Why Johnny Can't Encrypt*, modern PGP tools are still unusable for the masses. We finish with a discussion of pain points encountered using Mailvelope, and discuss what might be done to address them in future PGP systems.

Author Keywords

SoK: Why Johnny Can't Fix PGP Standardization

Harry Halpin
harry.halpin@inria.fr
Inria
Paris, France

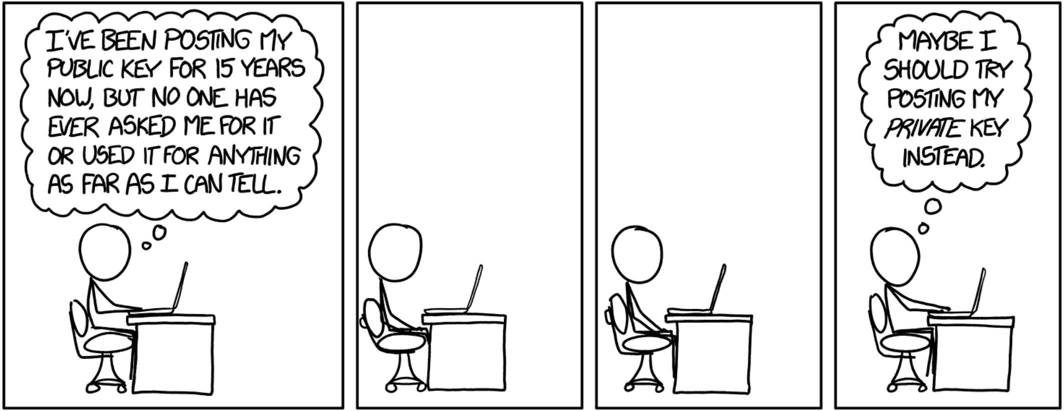
ABSTRACT

Pretty Good Privacy (PGP) has long been the primary IETF standard for encrypting email, but suffers from widespread usability and security problems that have limited its adoption. As time has marched on, the underlying cryptographic protocol has fallen out of date insofar as PGP is unauthenticated on a per message basis and compresses before encryption. There have been an increasing number of attacks on the increasingly outdated primitives and complex clients used by the PGP eco-system. However, attempts to update the OpenPGP standard have failed at the IETF except for adding modern cryptographic primitives. Outside of official standardization, Autocrypt is a "bottom-up" community attempt to fix PGP, but still falls victim to attacks on PGP involving authentication. The core reason for the inability to "fix" PGP is the lack of a simple developers created a new community effort called "Autocrypt" to address the underlying usability and key management issues. This effort also introduces new attacks and does not address some of the underlying cryptographic problems in PGP, problems that have been addressed in more modern protocol designs like Signal or IETF Message Layer Security (MLS). After decades of work, why can't the OpenPGP standard be fixed?

First, we start with the history of standardization of OpenPGP in Section 2. We consider the PGP protocol itself according to the modern understanding of cryptography in Section 3, inspecting whether some original design choices still make sense in terms of the security and privacy properties a user would expect. Then we move to summarizing the numerous attacks on PGP that the design and incomplete standardization of PGP opens up in Section 4.

- Common mistakes:
  - Encrypt a message with the sender's public key
  - Send private key so that recipient can decrypt a message
- Oftentimes, study participants cannot send a PGP-encrypted e-mail after 45min

Problem 1: Usability


I'VE BEEN POSTING MY PUBLIC KEY FOR 15 YEARS NOW, BUT NO ONE HAS EVER ASKED ME FOR IT OR USED IT FOR ANYTHING AS FAR AS I CAN TELL.
MAYBE I SHOULD TRY POSTING MY PRIVATE KEY INSTEAD.

(Public Key)

Problem 2: Key compromise

- Suppose (encrypted) communications between Alice and Bob are recorded by the “bad guys”
  - criminals, competitors, etc
- Later, Bob’s computer is stolen by the same bad guys
- Or just broken into
  - Virus, trojan, etc

All of Bob’s key material is recovered

The bad guys can...

- Decrypt past messages
- Learn their content
- Learn that Alice sent them
- And have a mathematical proof they can show to anyone else!

How private is that?

What went wrong?

- Bob’s computer got stolen?
- How many of you have never...
  - Left your laptop unattended?
  - Not installed the latest patches?
  - Run software with a remotely exploitable bug?
- What about your friends?

What really went wrong

- PGP creates lots of incriminating records:
  - Key material that decrypts data sent over the public Internet
  - Signatures with proofs of who said what
- Alice had better watch what she says!
  - Her privacy depends on Bob's actions

Module outline

11 Application layer security

12 SSH

13 PGP

14 OTR

15 Signal

Casual conversations

- Alice and Bob talk in a room
- No one else can hear
  - Unless being recorded
- No one else knows what they say
  - Unless Alice or Bob tells them
- No one can prove what was said
  - Not even Alice or Bob

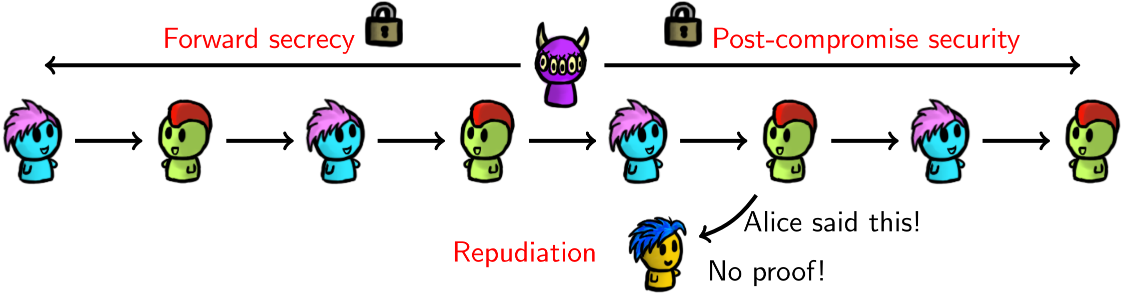
These conversations are “off-the-record” (OTR)

We like off-the-record conversations

- Legal support for having them
  - Illegal to record other people's conversations without notification
- We can have them over the phone
  - Illegal to tap phone lines
- But what about over the Internet?

What do we want to achieve?

- (Perfect) Forward secrecy: a key compromise does not reveal past communication.
- Post-compromise security ~~Backwards secrecy~~ ~~Future secrecy~~: a key compromise does not reveal future communication.
- Repudiation (deniable authentication): authenticated communication, but a participant cannot prove to a third party that another participant said something.



Repudiation (Deniable authentication)

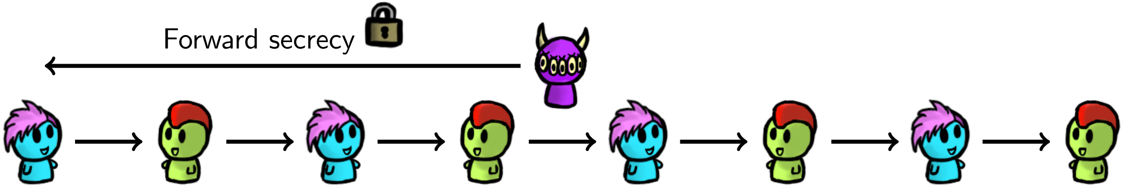
We want authentication with repudiation (deniable authentication).

Q: What do we use for this?

- Ⓐ Hash functions
- Ⓑ Checksums
- Ⓒ MACs
- Ⓓ Digital Signatures

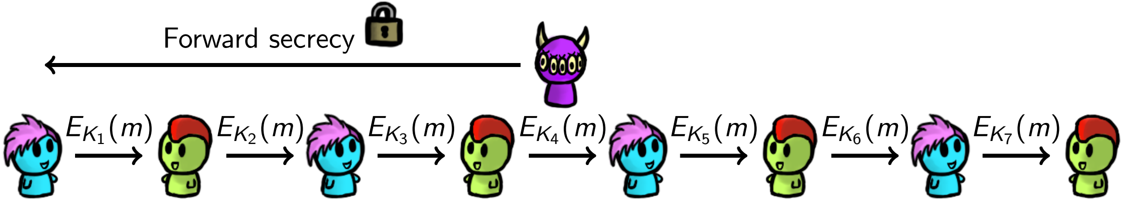
A: Message Authentication Codes (MACs)! Signatures provide non-repudiation, which is great for signing contracts but undesirable for private conversations.

(Perfect) Forward Secrecy



- Future key compromises should not reveal past communication
- Use secret-key encryption with a short-lived key (a session key)
- The session key is created by a modified Diffie-Hellman protocol
- Discard the session key after use:
    - Securely erase it from memory (and everywhere possible)
- Use long-term keys only to authenticate the Diffie-Hellman protocol messages only

Idea for Forward Secrecy:



Forward secrecy

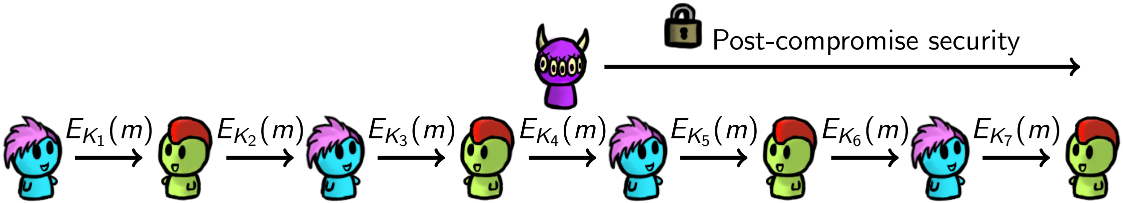
$E_{K_1}(m)$ $E_{K_2}(m)$ $E_{K_3}(m)$ $E_{K_4}(m)$ $E_{K_5}(m)$ $E_{K_6}(m)$ $E_{K_7}(m)$

- Future key compromises should not reveal past communication

Q: Idea: what if session keys are hashes of the previous key?
$K_1 \to K_2 = H(K_1) \to K_3 = H(K_2) \to \dots$

A: This should work (there are better ways of doing this, though)
What if a message arrives out of order?

Post-Compromise Security



Post-compromise security

$E_{K_1}(m)$ $E_{K_2}(m)$ $E_{K_3}(m)$ $E_{K_4}(m)$ $E_{K_5}(m)$ $E_{K_6}(m)$ $E_{K_7}(m)$

- Past key compromises should not compromise the security of future sessions

Q: Idea: what if session keys are hashes of the previous key?
$K_1 \rightarrow K_2 = H(K_1) \rightarrow K_3 = H(K_2) \rightarrow \ldots$

A: Eve can still compute all future keys!

- So what can we do?
  - Regularly replace potentially compromised session keys with new key material

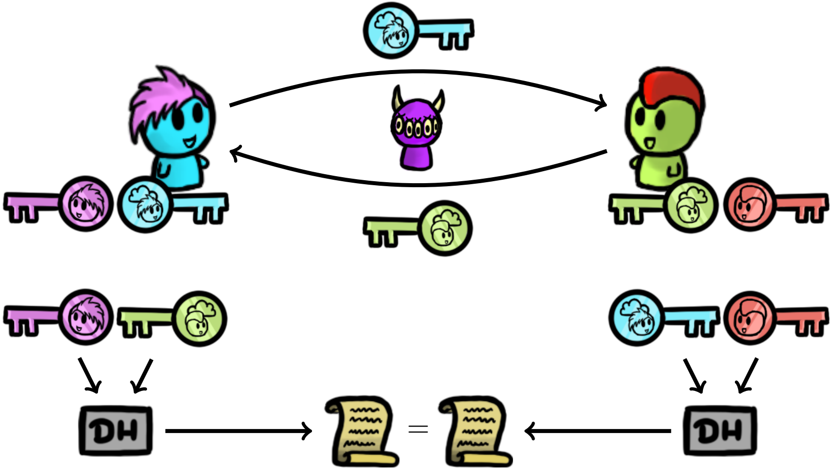
Cryptographic Ratchets

- A ratchet is a mechanical device that moves in one direction and cannot move backwards.
- Example of cryptographic ratchet:

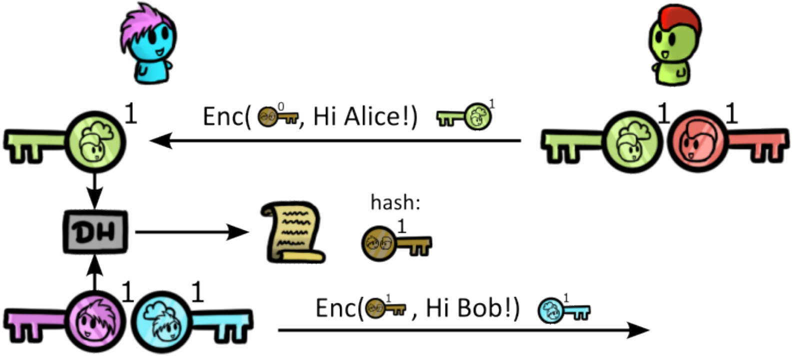
$$K_1 \to K_2 = H(K_1) \to K_3 = H(K_2) \to \ldots \tag{1}$$

- We want something that provides both *forward secrecy* and *post-compromise security*.
- We will see the OTR ratchet, which uses Diffie-Hellman.
- The signal protocol uses a more advanced version of this ratchet, which we will not see (Whatsapp also uses this now)

Recall: Diffie-Hellman

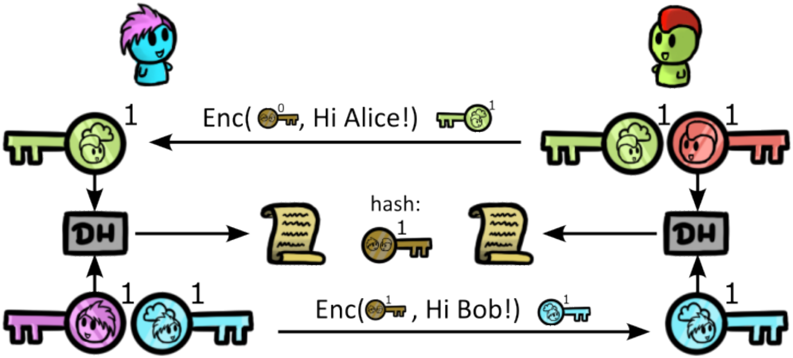


OTR’s DH Ratchet (visually)



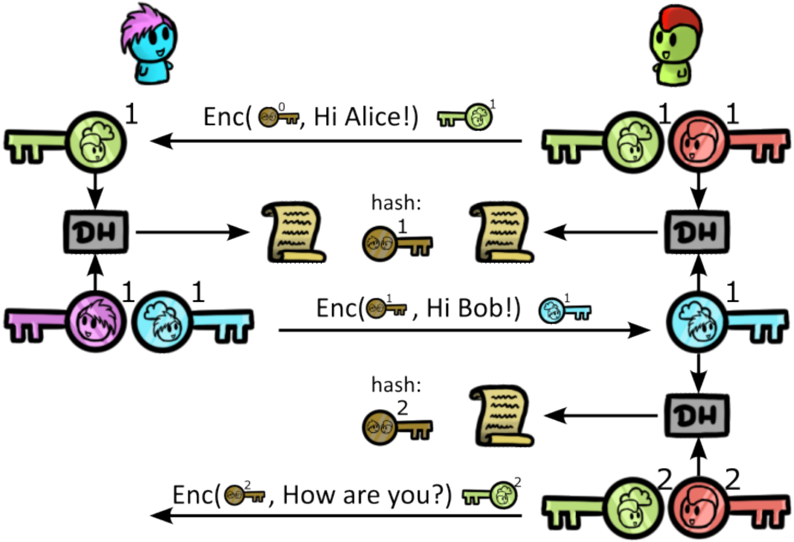
Q: How does Bob recover the message?

OTR's DH Ratchet (visually)

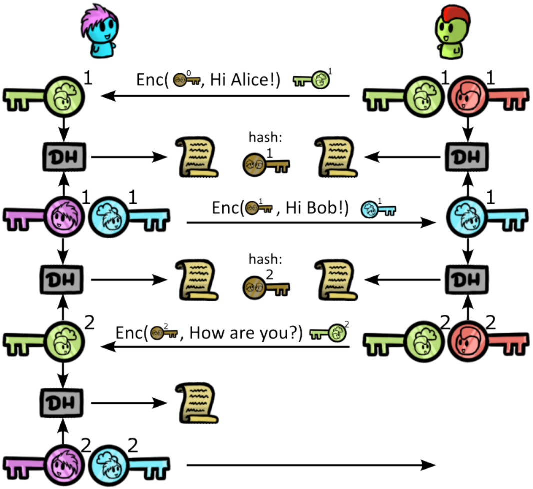


Q: Following this logic, how does Bob reply to Alice?

OTR's DH Ratchet (visually)



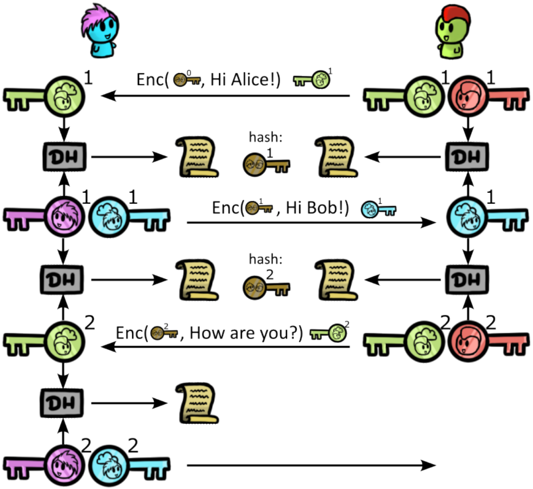
OTR's DH Ratchet (visually)



Q: What happens if Eve learns a private key? (e.g., ^{1})

A: She can decrypt "Hi Bob!" and "How are you?"

OTR's DH Ratchet (visually)



- Session keys only roll forward with interactive replies
- If Alice sends multiple messages but Bob takes a long time to reply, multiple messages get encrypted with the same key!
- Forward secrecy is only partially provided

Using these techniques

Using forward secrecy and deniable authentication, we can make our online conversations more like face-to-face "off-the-record" conversations.

But there is a wrinkle:

- These techniques require the parties to communicate interactively
- This makes them unsuitable for email
- But they're still great for instant messaging!

Module outline

11 Application layer security

12 SSH

13 PGP

14 OTR

15 Signal

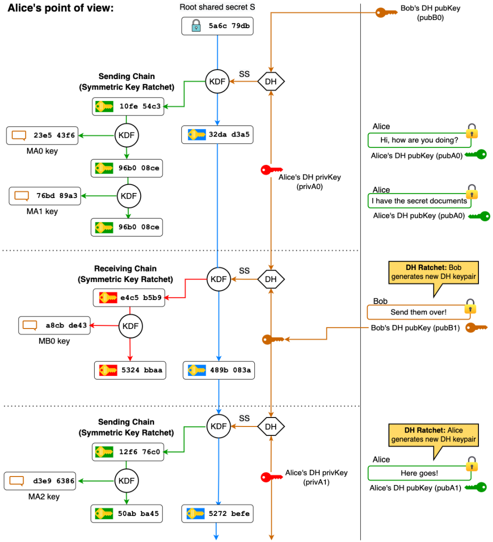
Signal Protocol

- Signal is an app for iOS, Android, and Chrome
  - Original protocol based on OTR and used for encrypted SMS (e.g., Google Messages)
- The Signal Protocol is now used by other apps like WhatsApp
  - Also optionally in Facebook Messenger and Skype
  - Why on Earth would you like to always keep your conversations private, right? :-)

Signal Protocol

- Provides forward secrecy
  - Similar to OTR, uses a "ratchet" technique to constantly rotate session keys
- Provides post-compromise security
  - A leak of past or long-term keys will be healed by introducing new DH ratchet keys
- Provides improved deniability
  - Uses "Triple Diffie-Hellman" deniable authenticated key exchange
- Supports out-of-order message delivery
  - Users can store per-message keys until late messages arrive
- Uses a double ratchet (assymetric and symmetric ratchets) that
  - Generates ephemeral per-message keys
  - Tolerates message loss/re-ordering

Exchanging Messages (the full picture)



Alice
Hi, how are you doing?
Alice's DH pubKey (pubA0)

Alice
I have the secret documents
Alice's DH pubKey (pubA0)

Bob
Send them over!
Bob's DH pubKey (pubB1)

Alice
Here goes!
Alice's DH pubKey (pubA1)

The double ratchet

- Just kidding, you don't need to know this, but it's very well explained on the Signal website: https://signal.org/docs/specifications/doubleratchet/
- You simply have to know that Signal protocol provides forward secrecy, post-compromise security and deniable authentication, building upon OTR.

Recap: cryptography use cases (Pt. II)

- SSH:
  - the connection protocol
  - pros/cons of each client authentication option
  - SSH port forwarding
- PGP
  - hybrid encryption, digital signatures
  - PGP → OpenPGP: offers many options, it's complex
  - web of trust (understand how it works)
  - issues: usability, incriminating records
- OTR:
  - forward secrecy, post-compromise security, repudiation
  - OTR DH Ratchet (good for interactive communication)
- Signal:
  - builds upon OTR (more complicated ratchet)
  - provides forward secrecy, post-compromise security, repudiation, supports out-of-order messages

CS 458 / 658: Computer Security and Privacy

Module 5 – Security and Privacy of Internet Applications

Part 4 – Privacy-Enhancing Technologies — PETs

Fall 2022

Module outline

16 Privacy and Anonymity

17 Mixes

18 Attacks on Mixes

19 A Brief History of Remailers

20 Tor

21 Private information retrieval (PIR)

What is Privacy?

- Recall from Module 1: privacy is "informational self-determination"
- In other words: you get to control information about you
- Control means: who gets to see it, who gets to use it, what they can use it for, etc.

Q: What does privacy mean to you? What would you draw if you had to draw "privacy"?

- Paper on PoPETs 2018: Turtles, Locks, and Bathrooms: Understanding Mental Models of Privacy Through Illustration
- Asked people of different ages in the US to draw a diagram on what privacy means to them, and here are a few illustrations.

Privacy as turtles

Fig. 63. “It's a turtle huddled up inside its shell.” By John

PEARL OYSTERS HAVE SOMETHING VALUABLE
TO PROTECT - THE PEARL.
THEY CAN DO SO BY SIMPLY 'CLOSING THE LID'
IF ONLY SAFEGUARDING THE DATA IN MY
LAPTOP WERE THAT SIMPLE!

Fig. 62. “Pearl oysters have something valuable to protect - the pearl. They can do so by simply ‘closing the lid.’ If only safeguarding the data in my laptop were that simple!” By Sharon, age 25.

Fig. 40. “A shield that protects me.” By HAP, age 24

[0] All pictures from the PoPETs'18 paper

Privacy as locks

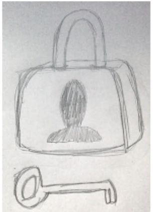


Fig. 10. "To me, privacy is fundamentally about feeling secure. Having the ability to control who has access to me, and to my information, makes me feel like I can control my privacy." By CJ, age 33

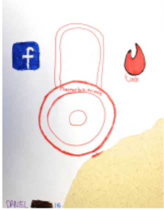


Fig. 11. By Daniel, age 16

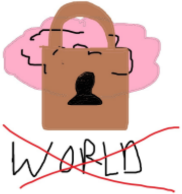


Fig. 33. "Privacy means that the thoughts in my brain are locked away. What I know does not have to go into the world, which I put an X over." By Thomas, age 19

[0] All pictures from the PoPETs'18 paper

Privacy as bathrooms

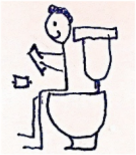


Fig. 23. "This is me enjoying my privacy. This is the only time during the day, were I am truly alone and nothing bothers me. No man no children no dogs." By Cindy, age 54

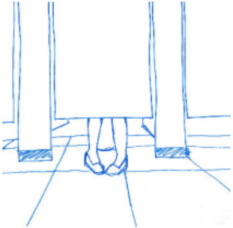


Fig. 38. By Rachel, age 20

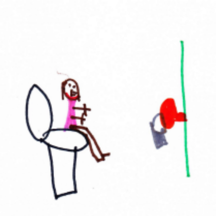


Fig. 24. "No one come in when I am in the bathroom!" By Sydney, age 7

[0] All pictures from the PoPETs'18 paper

Privacy as filters

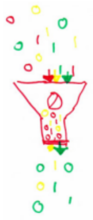


Fig. 45. "Green data (non-private) goes through; red does not (private data). Some yellow goes through (ambiguous)." By Ryan, age 36

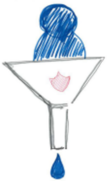


Fig. 58. "Privacy is to me the ability to filter and control the information relevant to you that you release into the world (and having some confidence in the ability of the status of such information as private)." By Isadora, age 20

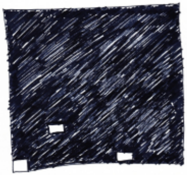


Fig. 74. "Privacy means my life is a black box, except for the items I choose to share with others." By Lauren, age 32

[0] All pictures from the PoPETs'18 paper

Privacy as controls

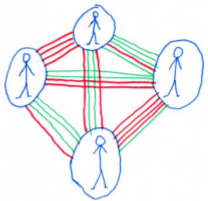


Fig. 46. 'Privacy is a network: I share what I want with whom I want and trust and what matches with those in the network, and don't share with those I don't want and trust to share with. Green = share. Red = don't." age 20s

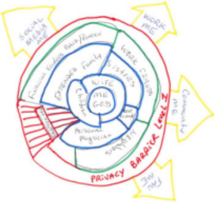


Fig. 47. "I give/receive based on my level of trust. Occasionally, I do not share with those I trust (i.e., my exception jail) as I do not trust what they will do with a specific piece of information. I accept that I must have a public persona." By Jim, age 51

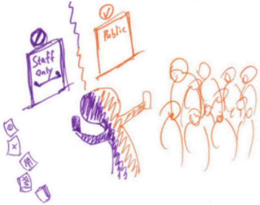


Fig. 55. "There are bright sides, and there are dark sides. Some of them we'd love to share; some we don't, and they are called 'privacy."' By Evan, age 21

[0] All pictures from the PoPETs'18 paper

Privacy as tools

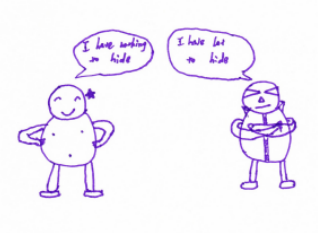


Fig. 54. By Lidong Wei

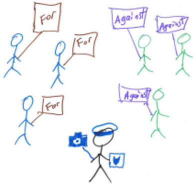


Fig. 30. "People should be able to express their views without surveillance & infiltration by the police." By anonymous, age "old"

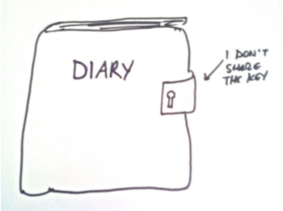


Fig. 28. "The picture is of a diary that has a mechanism to keep it shut. There is no key available within the drawing, so that no one can open it. If no one can open it, privacy remains intact." By Karen, age 43

[0] All pictures from the PoPETs'18 paper

So, what is privacy?

- Privacy means something different to each person ("Informational self-determination")
- There are two "types" of information that could be privacy-sensitive:
  - Data: refers to contents of messages, contents of a database, etc.
  - Meta-data: any other information that is not data; for example, in communications, meta-data includes:
    - Who communicates with whom?
    - What time does Alice communicate with Bob?
    - How often does Alice communicate with Bob?
    - Where does Alice communicate from?
    - Does Alice communicate with anyone at all?
    - ...
- We can hide data using cryptography, but sometimes we need to leak some data to a potential adversary to get some utility from a service (many services have a privacy-utility trade-off).
- Protecting meta-data requires more than cryptography.

What we will cover

We need Privacy-Enhancing Technologies (PETs) to control data leakage, as well as to protect meta-data!

- We'll only cover PETs that are related to two aspects of privacy:
  - Anonymity in communications (privacy as masks): how to hide who communicates with whom; we'll see remailers (mixes) and Tor.
  - Data minimization (privacy as filters): how to achieve a functionality while minimizing the amount of data collected; we'll see Private Information Retrieval (PIR)

Kerckhoff's principle, again

Remember this from the first lecture of this module?

Kerckhoff's principle

a cryptosystem should be secure, even if everything about the system, except the key, is public knowledge.

Shannon's maxim

one ought to design systems under the assumption that the enemy will immediately gain full familiarity with them.

We also need to keep these into account when designing PETs!
We assume the adversary knows how the technology works, and we still want to get privacy

Anonymity

- Anonymity refers to "the state of not being identifiable within a set of subjects, the anonymity set"[1].
  - A sender may be anonymous within a set of potential senders: the sender anonymity set.
  - Same for recipients.
- Pseudonymity is the use of *pseudonyms* as IDs.

[1] from *Pfitzman et al*, "Anonymity, unobservability, and pseudonymity—a proposal for terminology"

Anonymity

We can place transactions (online/offline) on a continuum according to the level of *nymity* they represent:

- **Verinymity**: almost unique information (Government ID, SIN, credit card number, address)
- **Persistent pseudonyms**: a “handle” or “nickname” that you use persistently by the same person (Twitter/Instragram accounts, posting blogs under a pseudonym, etc.)
- **Linkable anonymity**: prepaid phone cards, loyalty cards
- **Unlinkable anonymity**: cash payments, remailers, Tor

Anonymous Communication Systems

Anonymous communication systems are typically classified into:

- High-latency anonymous communication systems:
  - Provide protection against global passive adversaries
  - Have higher delays (fine for email)
  - We will see mixes (remailers).
- Low-latency anonymous communication systems:
  - Do not protect against global passive adversaries, but are good against local adversaries.
  - Have lower delays (fine for browsing)
  - We will see Tor

The latency of high-latency anonymous communication systems has decreased significantly, they are not really "high-latency" anymore.

Module outline

16 Privacy and Anonymity

17 Mixes

18 Attacks on Mixes

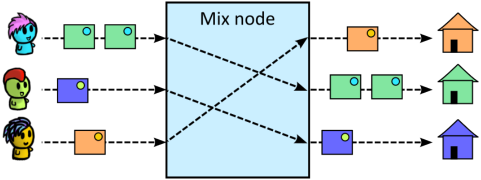
19 A Brief History of Remailers

20 Tor

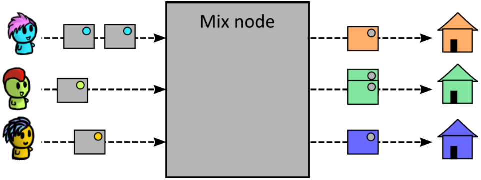
21 Private information retrieval (PIR)

Mixes: basic operations

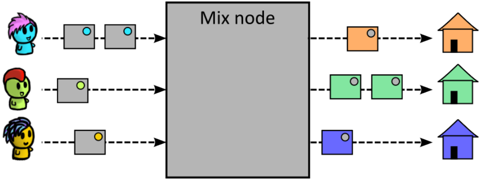
How do we provide anonymity?



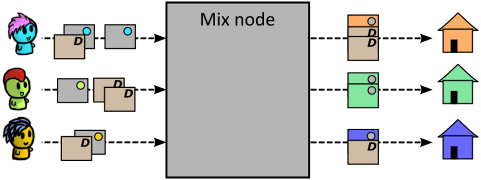
Delay messages!



Change appearance!

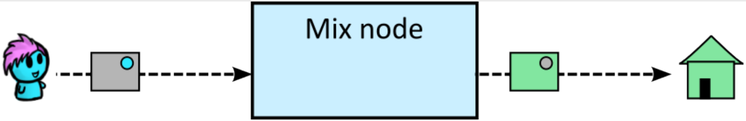


Add dummy traffic!



Operation 1: Changing Appearance

Q: How can we achieve this? (clue: we have some crypto tools!)



A:

- We can encrypt the output message with the Mix's key

⬜ = $E_{K_{mix}}($ 🟩 $)$

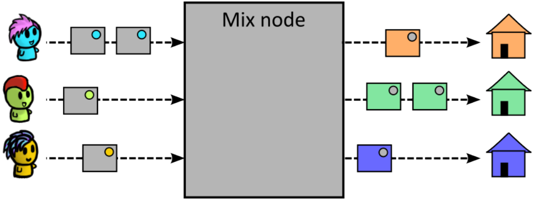
🟩 = $E_{K_{Bob}}(m)$

- This "layered encryption" concept is called *onion routing*, and we will see it later in Tor.

Operation 2: Delaying Messages

Q: How do we do this?

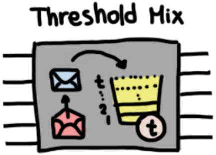
- Do we add a random delay to each message?
- Do we add a deterministic delay to each message?
- Do we add a constant delay to each message?



A: Yes. Yes. No. Deterministic delay: it's not constant, it depends on the arrival time and/or other messages. We will see some examples next!

Threshold and Timed Mixes

- Some popular mixes types are threshold and timed mixes.
- These mixes gather messages until a flushing condition triggers.
- When this condition happens, this marks the end of a round
  - Threshold mix: it gathers t messages, then it flushes them.
  - Timed mix: it gathers messages until a timer set to τ seconds expires, then it flushes them.

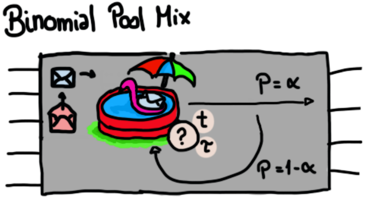


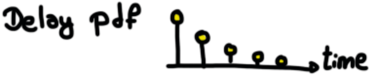
Q: Which of the two is better?

A: It depends... the threshold mix ensures a certain mixing size, the timed mix ensures a maximum message delay.

Pool Mixes

- When a (threshold/timed) mix keeps some messages inside after a round ends, it is called a *pool mix*.
- The binomial pool mix forwards each message with probability α, and keeps it inside the mix with probability $1 - \alpha$.





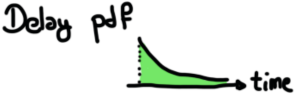
Q: What are the pros and cons of this?

A: Pros: more anonymity; cons: more delay

Continuous-time or Stop-and-Go (SG) Mixes

- Some mixes do not work on “batches” or “rounds”, and instead delay each message independently: these are called continuous-time mixes or Stop-and-Go (SG) mixes.
- Mixes that delay messages following an exponential distribution are very popular (Loopix, Nym).
- The user can choose the delay and include it in the message

Stop-and-Go (SG)
or Continuous Mixes
d
delay
1:26pm



Mixnets

- Sending messages through a single mix is not great

Q: Why?

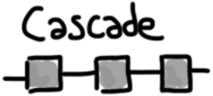
A: There’s a single point of failure, and the mix knows the message correspondence.

- We can chain mixes to create a mixnet.
- Mixnets have different topologies, depending on which nodes a message can travel between.

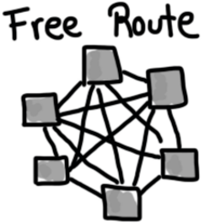
Mixnet Topologies

Let’s discuss pros and cons of each topology!

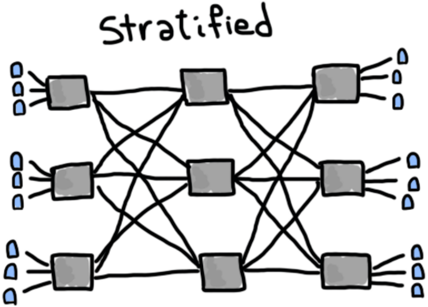
- Cascade: one after the other



- Freeroute: all of them are connected



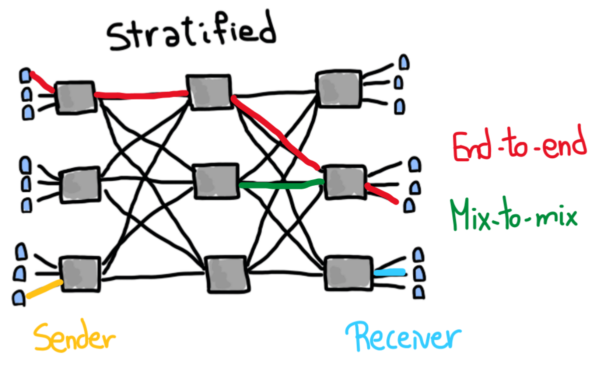
- Stratified: each layer is fully connected to the next layer



Operation 3: Dummy Messages

Q: Where do we add dummy traffic?

A: Anywhere! Everywhere!

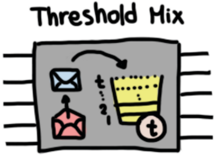


Recap (I)

Q: What are the three basic operations of a mix node to provide anonymity? Why is each operation important?

A: Change appearance, delay messages, add dummy traffic

Q: Threshold mixes: pros and cons of increasing the threshold t?



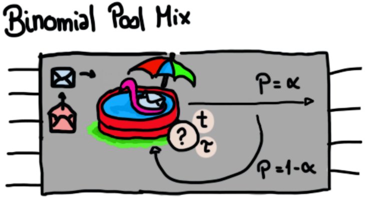
A: Increasing t improves anonymity but increases delay

Recap (II)

Q: Timed mixes: pros and cons of increasing the time τ?

A: Increasing τ improves anonymity but increases delay

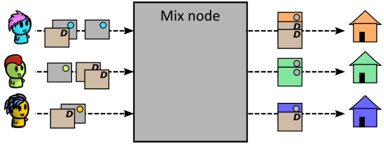
Q: Binomial pool mix: pros and cons of increasing the probability of forwarding a message α?



A: Increasing α decreases anonymity and delay

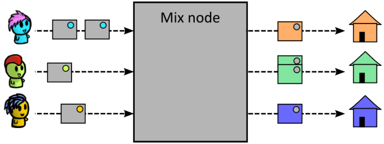
Recap (III)

Q: Dummy traffic: pros and cons of increasing the amount of dummy messages?



A: More dummies require more bandwidth, but increase anonymity

Q: What happens if the number of senders increases?



A: Depends on the actual mix/setting, but usually *anonymity loves company*. More people using the system usually improves its anonymity level.

Anonymity trade-offs

Anonymity has a cost. We can increase anonymity by:

- Adding more message delay
  - It has to be added “cleverly” (e.g., a constant delay does not work)
- Adding more dummy traffic
  - It has to be added “cleverly” (e.g., simulating real sending behavior)
- When the number of users increases
  - Effectiveness depends on the type of mix, the mix topology, etc.

Module outline

16 Privacy and Anonymity

17 Mixes

18 Attacks on Mixes

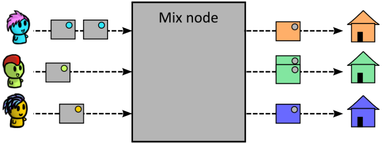
19 A Brief History of Remailers

20 Tor

21 Private information retrieval (PIR)

Attacks on Anonymous Communication Systems

Q: If you are an active adversary, how would you attack a mix? (e.g., a *threshold* mix)

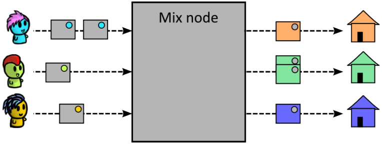


A: A possible attack, called the $n-1$ attack: Mallory sends all but one message (e.g., $t-1$ in this case). Mallory can identify her messages leaving the mix, so the remaining message has to be Alice's.

Any other ideas?

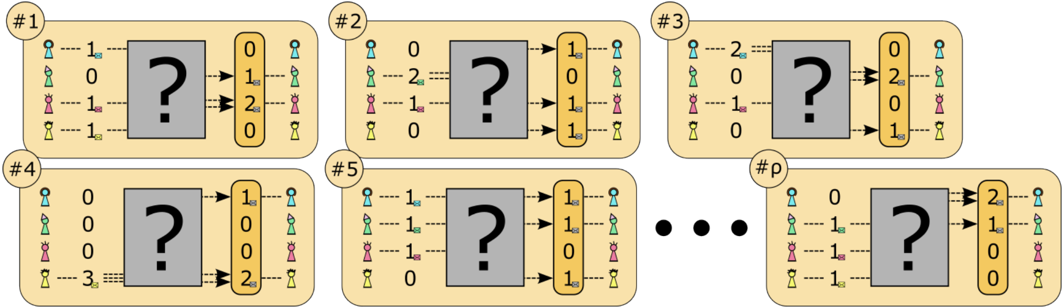
Attacks on Anonymous Communication Systems

Q: If you are a passive adversary, how would you attack a mix (e.g., a *threshold* mix)



A: We can exploit sending behavior (e.g., Alice sent two messages, so “mostly likely” they were for the same recipient?)
But we can't do much more here...

Attacks on Anonymous Communication Systems

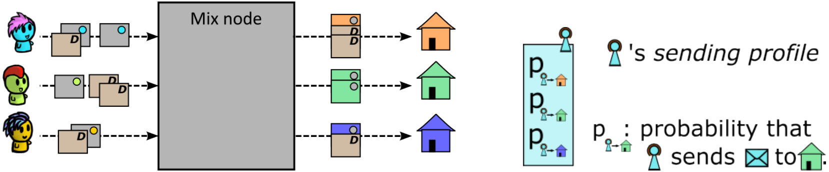


Q: What if we are a passive adversary observing the mix for a long time? Who is Alice (first sender) most-likely sending messages to?

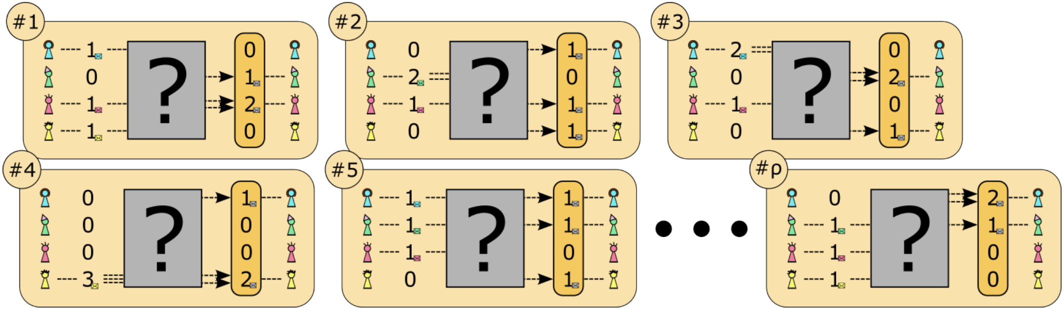
A: Probably the second receiver...

Long-term Profiling Attacks on Mixes

- A global passive adversary (Eve) observing incoming and outgoing messages for a single round in a mix will likely not learn a lot about the users.
- However, in the *long term*, some communication patterns will become more obvious, and this will enable long-term profiling attacks.
- The goal of a long-term profiling attack is to estimate the *sending profile* of a sender (i.e., which parties the sender communicates with, and how often).
- We will see the simplest example of this: the Statistical Disclosure Attack (SDA)

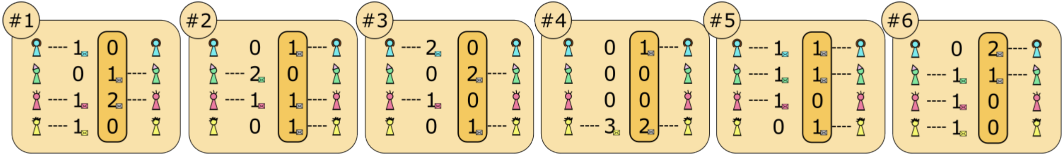


Attack Intuition

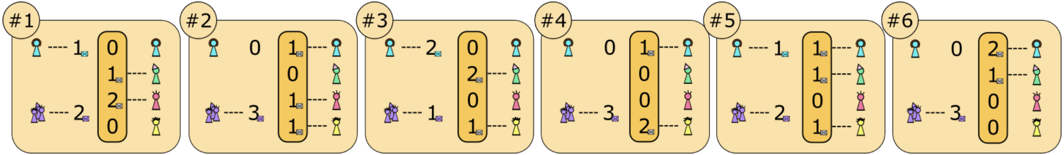


- Even sees many communication rounds. In this case, this is a threshold mix with $t = 3$.
- We're going to see the case where the adversary just wants to estimate Alice's sending profile.

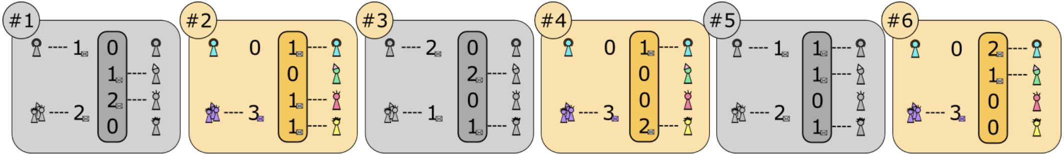
Attack Intuition



- Let's try to make this attack more "principled"
- Imagine we just want to estimate Alice's sending profile. We first separate Alice from the "others", also called the background traffic



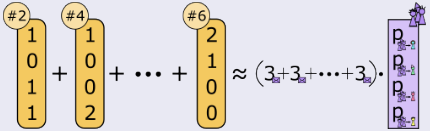
Attack Intuition



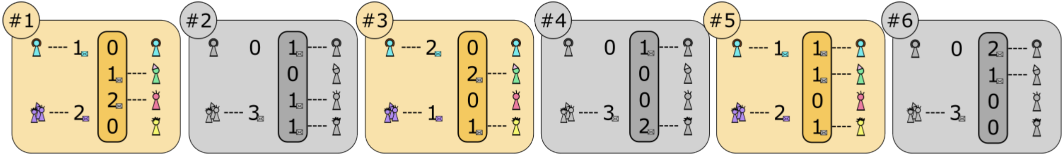
Q: Using the rounds where Alice does not participate, what's a very simple way of estimating the sending profile of the "background"?

A: We can average the outputs across rounds:

E[msgs received by j]=(msgs sent by bkg)· Pr(bkg sends to j)



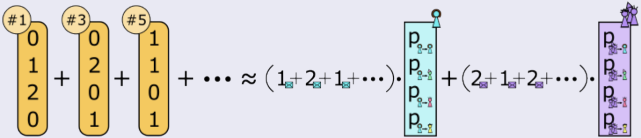
Attack Intuition



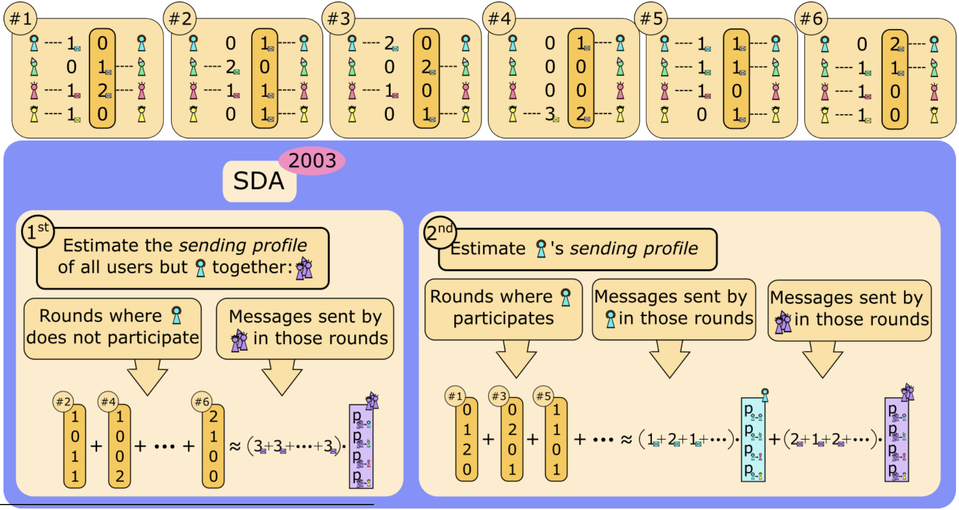
Q: Now we have the background profile; using the rounds where Alice *does* participate, what's a very simple way of estimating Alice's sending profile?

A: We can do the same average-case analysis, but now both Alice and the background contribute to the average:

E[msgs received by j]=(msgs sent by bkg)· Pr(bkg→ j)+(msg sents by Alice)· Pr(Alice→ j)



Statistical Disclosure Attack (SDA)



SDA 2003

1st Estimate the *sending profile* of all users but together:

Rounds where does not participate

Messages sent by in those rounds

#2 1 0 1 1 + #4 1 0 0 2 + ··· + #6 2 1 0 0 ≈ (3+3+···+3)· p p p p

2nd Estimate 's *sending profile*

Rounds where participates

Messages sent by in those rounds

Messages sent by in those rounds

#1 0 1 2 0 + #3 0 2 0 1 + #5 1 1 0 1 + ··· ≈ (1+2+1+···)· p p p p + (2+1+2+···)· p p p p

[1] Danezis, George. "Statistical disclosure attacks."

Practice: SDA

Q: In the example: can you tell who's Alice's best friend? What is the estimation of Alice's sending profile according to SDA?

A: The background profile:

$$p_{bkg} = [4, 1, 1, 3]/9 = \left[\frac{4}{9}, \frac{1}{9}, \frac{1}{9}, \frac{1}{3}\right] \approx [0.44, 0.11, 0.11, 0.33]$$

Alice's profile:

$$p_{alice} = \frac{1}{4} \cdot ([1, 4, 2, 2] - 5 \cdot p_{bkg}) = \left[-\frac{11}{36}, \frac{31}{36}, \frac{13}{36}, \frac{3}{36}\right] \approx [-0.30, 0.86, 0.36, 0.08]$$

Note that the estimate adds up to 1! But we could have negative values, because this is just an *estimate* of Alice's sending profile (it's an unbiased estimate whose variance goes to 0 as the number of observed rounds goes to ∞, so if we increase the number of observed rounds, eventually we won't have any negative entries)

This profile means that Alice is "most likely" not sending messages to the first receiver, and "most likely" sending messages to the second receiver

Concluding notes on attacks

- This is the simplest statistical attack: there are more clever ways of estimating sending profiles given the observations
- You need to know how to attack a mix (active and passive attacks), and why the attacks work
- You need to know what we can do to improve the protection against attacks

Module outline

16 Privacy and Anonymity

17 Mixes

18 Attacks on Mixes

19 A Brief History of Remailers

20 Tor

21 Private information retrieval (PIR)

A Brief History of Remailers: Type 0

In the 1990s, there were very simple ("type 0") remailing services, the best known being anon.penet.fi (1993–1996)

Here is how it worked:

- Send email to `anon.penet.fi`
- It is forwarded to your intended recipient
- Your "From" address is changed to `anon43567@anon.penet.fi` (but your original address is stored in a table)
- Replies to the anon address get mapped back to your real address and delivered to you
- ≈ 10 000 emails per day (≈ 700 000 users)

anon.penet.fi

This works, as long as:

- No one's watching the Internet connections to or from `anon.penet.fi`
- The operator of `anon.penet.fi`, the machine (hardware), and the software all remain trustworthy and uncompromised
- The mapping of anon addresses to real addresses is kept secret

Unfortunately, a lawsuit forced Julf (the operator) to turn over parts of the list, and he shut down the whole thing, since he could no longer legally protect it

Type I remailers

Cypherpunk (type I) remailers removed the central point of trust

- Messages are now sent through a “chain” of several remailers, with dozens to choose from
- Each step in the chain is encrypted to avoid observers following the messages through the chain
- Remailers also delay and reorder messages

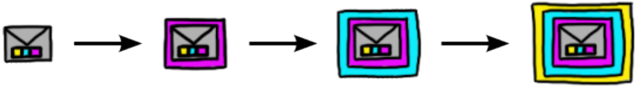
Support for pseudonymity is dropped: no replies!

Nym servers / pseudonymous remailers

How to do replies? (i.e., recovering pseudonymity)

- “nym servers” mapped pseudonyms to “reply blocks” that contained a nested encrypted chain of type I remailers.
- Alice picks a list of nym servers

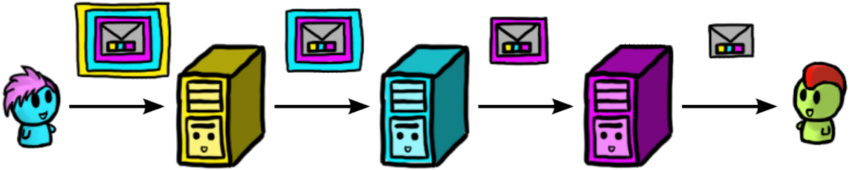
- Then, Alice builds her message using layered encryption
- The message contains a chain of reply blocks



Nym servers / pseudonymous remailers

How to do replies? (i.e., recovering pseudonymity)

- "nym servers" mapped pseudonyms to "reply blocks" that contained a nested encrypted chain of type I remailers.
- Alice picks a list of nym servers, and builds her message using layered encryption
- The message contains a chain of reply blocks
- Bob replies by attaching his response to the end of the reply blocks



Type II remailers

Mixmaster (type II) remailers appeared in the late 1990s

- Constant-length messages to avoid an observer watching “that big file” travel through the network
- Protections against replay attacks
- Improved message reordering

Requires a special email client to construct the message fragments

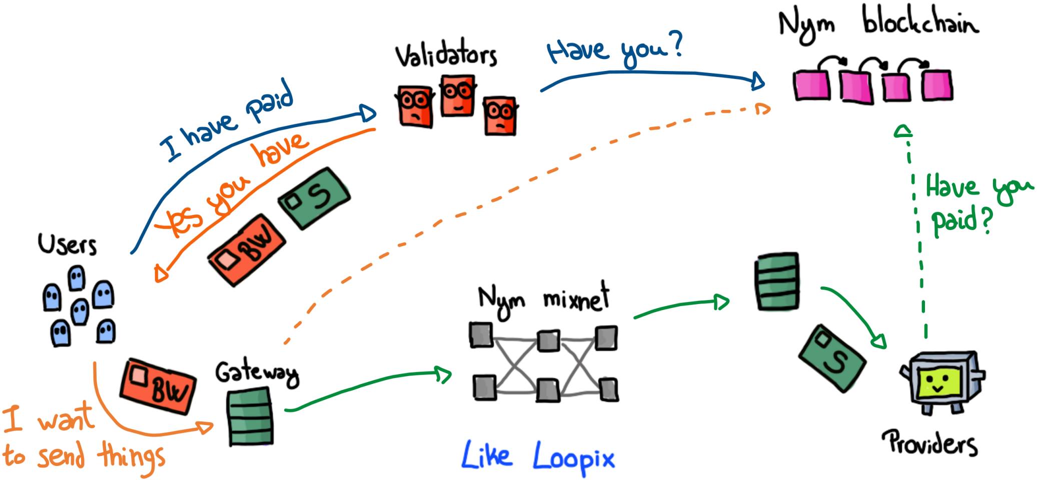
Type III remailers

Mixminion (type III) remailer appears in the 2000s

- Native (and much improved) support for pseudonymity
  - No longer reliant on type I reply blocks
  - Instead, relies on mix networks
- Improved protection against replay and key compromise attacks

But it's not very well deployed or mature, i.e., "you shouldn't trust Mixminion with your anonymity yet"

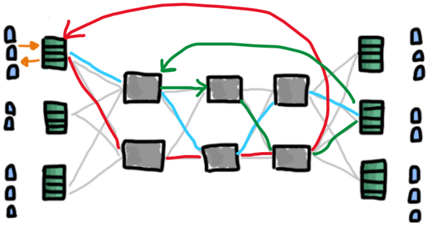
The Nym Network [Claudia Diaz, Harry Halpin, and Aggelos Kiayias (2021)]



The Nym Network

- Nym is a new privacy infrastructure that has three main components:
  - The Nym mixnet: a 3-layer stratified mixnet
  - The Nym credentials
  - The Nym token
- The Nym mixnet is largely based on Loopix:

- Stratified topology, 3 layers
- Continuous-time mixes (exponential delay)
- UDP, no circuits
- Real messages and drop dummies (in blue)
- Loop dummies (in red)
- Mix loop dummies (in green)
- Low latency (seconds!)



- Gateways (the green boxes)

Module outline

16 Privacy and Anonymity

17 Mixes

18 Attacks on Mixes

19 A Brief History of Remailers

20 Tor

21 Private information retrieval (PIR)

Tor - purpose

Tor is a successful privacy enhancing technology that works at the transport layer with ≈2 million daily users

Why do we need Tor when we have TLS?

- TLS protects data.
- We also want to protect metadata about the communication: e.g., IP addresses, browser fingerprints.

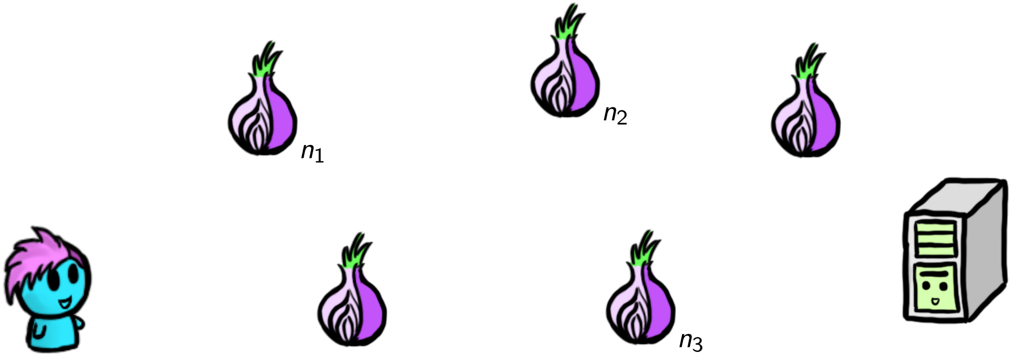
Tor is a low-latency anonymous communication system

- Tor has about 7 000 nodes scattered around the Internet; these are also called Onion Routers

Tor makes internet browsing unlinkably anonymous. But Tor does not (and cannot) hide the existence of the transaction (website visit) altogether

Tor: Building a Circuit (I)

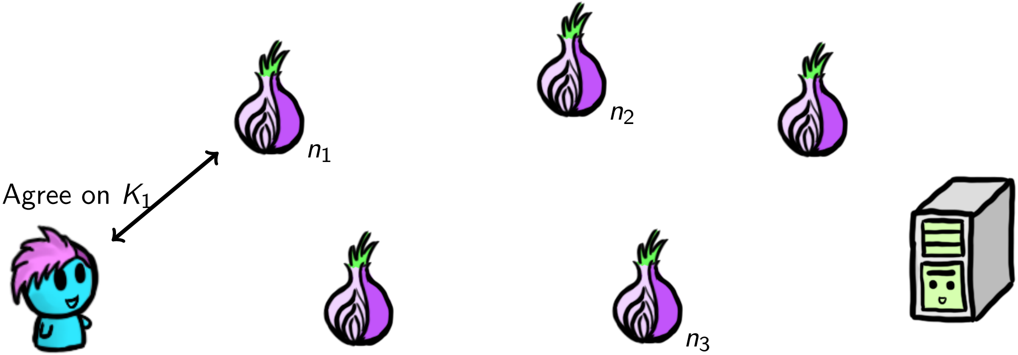
Alice wants to connect to a server without revealing her IP address



Alice has a global view of available Onion Routers (and their verification keys!)

Tor: Building a Circuit (II)

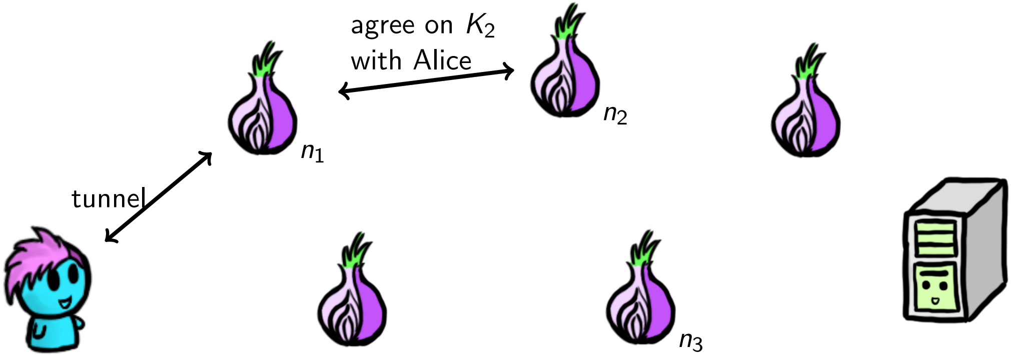
Alice picks one of the Tor nodes (n_1) and uses public-key cryptography to establish an encrypted communication channel to it (much like TLS)



The result is a secret key K_1 shared by Alice and n1

Tor: Building a Circuit (III)

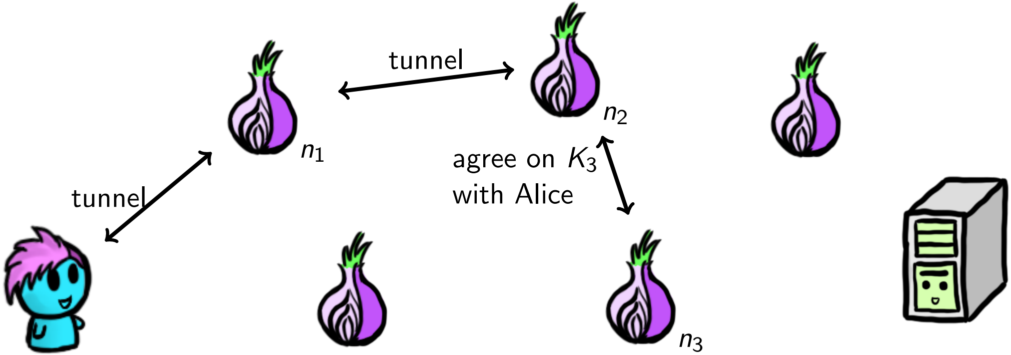
Alice tells n_1 to contact a second node (n_2), and establishes a new encrypted communication channel to n_2, tunneled within the previous one to n_1



The result is a secret key K_2 shared between Alice and n_2, which is unknown to n_1

Tor: Building a Circuit (IV)

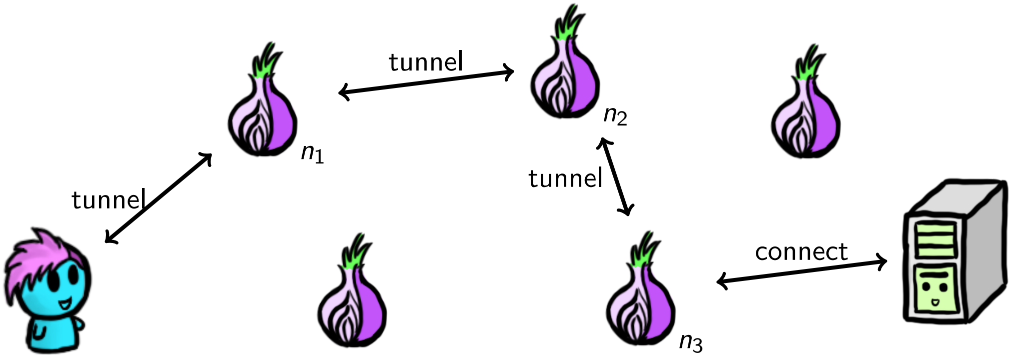
Alice tells n_2 to contact a third node (n_3), establishes a new encrypted communication channel to n_3, tunneled within the previous one to n_2.



The result is a secret key K_3 shared between Alice and n_3, which is unknown to n_1 and n_2

Tor: Building a Circuit (V)

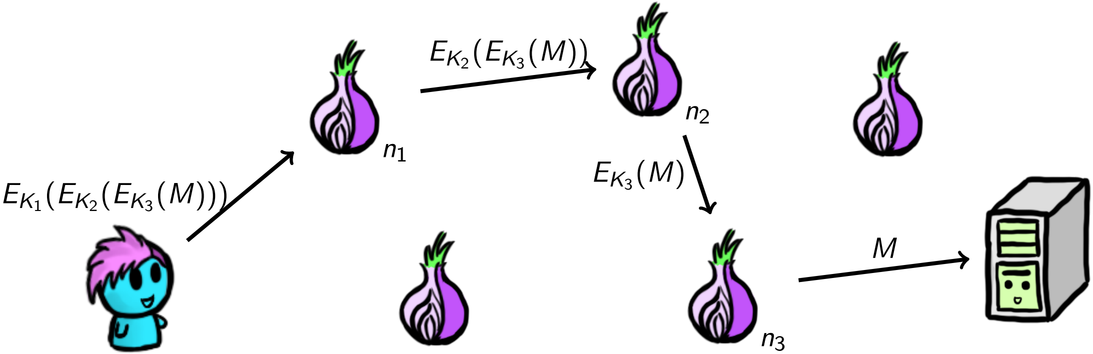
… And so on, for as many steps as she likes (usually 3) …



Alice tells the last node (within the layers of tunnels) to connect to the website

Sending messages with Tor

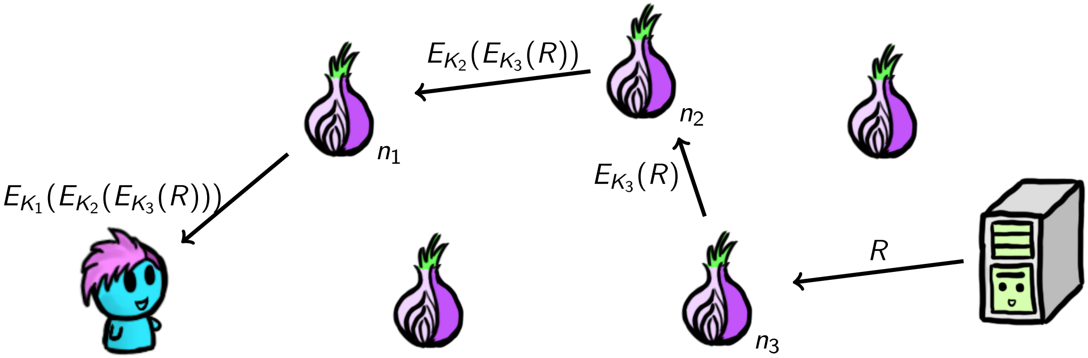
Alice encrypts her message “like an onion”; each node peels a layer off and forwards it to the next step



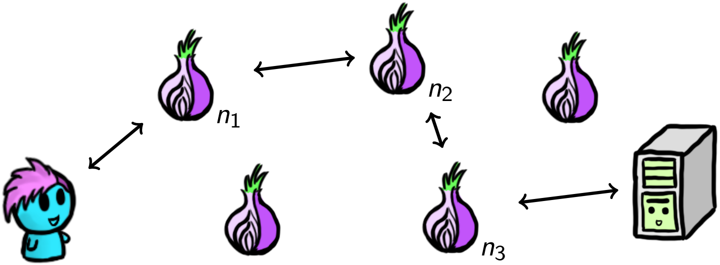
If connecting to a web server, M is encrypted (e.g., TLS)

Replies in Tor

The server replies with R, sending it back to n_3. The nodes encrypt the message back and Alice decrypts all the layers.

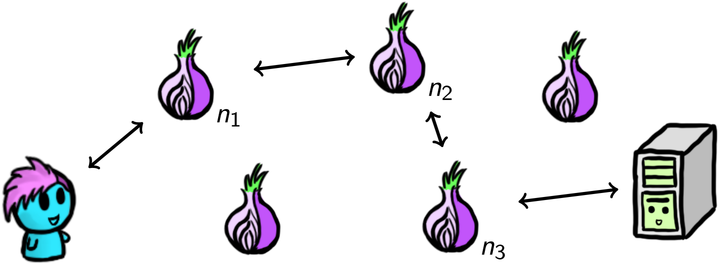


Who knows what?



- Notice that node n_1 knows that Alice is using Tor, and that her next node is n_2, but does not know which website Alice is visiting
- Node n3 knows some Tor user (with previous node n_2) is visiting a particular website, but doesn't know who
- The website itself only knows that it got a connection from Tor node n_3

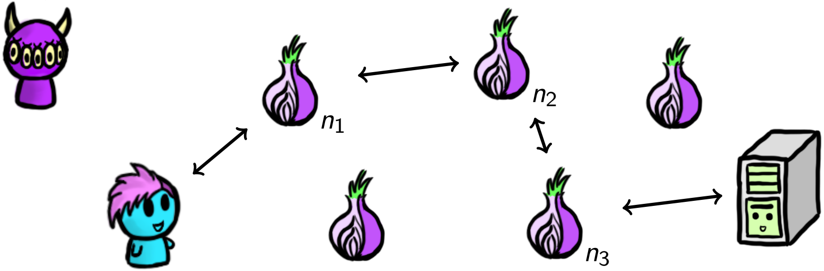
Some questions



Q: Why must Alice choose all nodes, instead of letting each node pick the next one?

A: A malicious node would pick another malicious node. The user must have the ability to choose the nodes

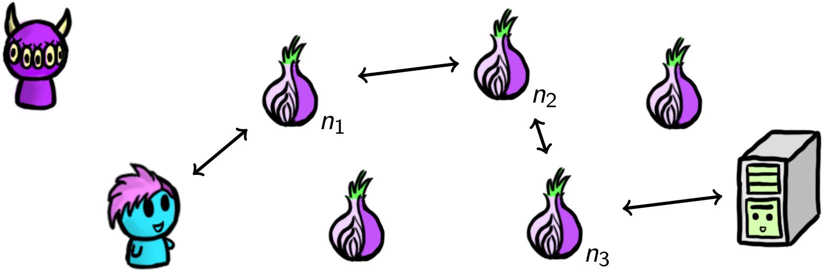
Some questions



Q: Why happens if Eve can inspect all network links? (a global passive adversary)

A: Tor does not protect against a global passive adversary. The adversary could de-anonymize Alice.

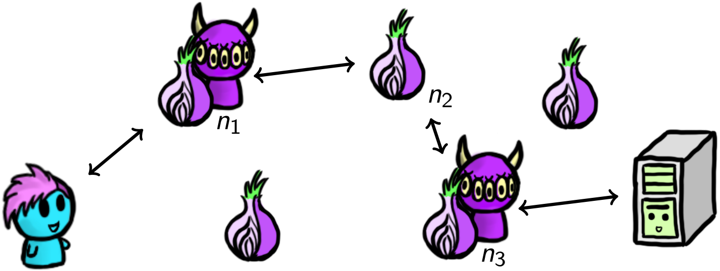
Some questions



Q: What happens when Eve can inspect the incoming and outgoing traffic of a *single* node?

A: Alice is probably good

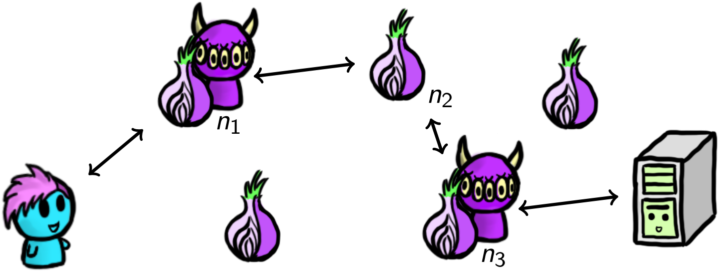
Some questions



Q: What happens when Eve can inspect the incoming and outgoing traffic of the first and last nodes?

A: Traffic correlation attacks can easily de-anonymize Alice

Some questions

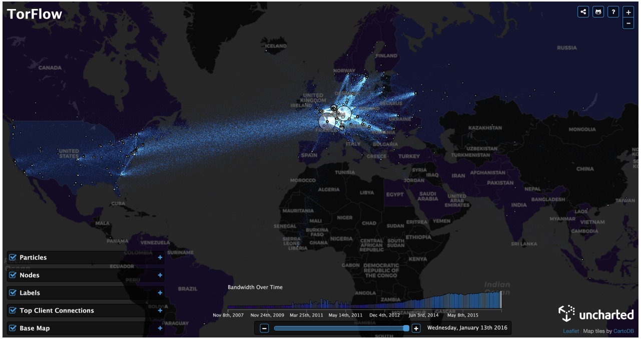


Q: Why do we usually pick 3 nodes?

A: It's a sweet spot between privacy and latency. More nodes usually do not provide more anonymity.

Path selection

- We want to avoid a global passive adversary: choose nodes in different ISPs/countries
- How concentrated is the geographical distribution of Tor relays?

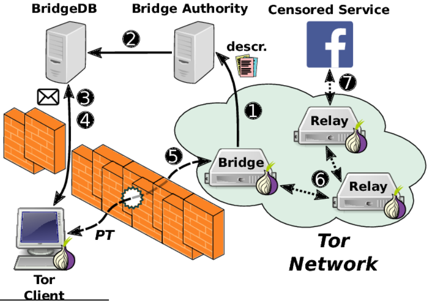


Path selection

- Path selection algorithms can help
    - With anonymity: by picking nodes that are in different countries/ISPs
    - With performance: latency is affected by this
- Don't forget that countries can collaborate as well
- We cannot use defenses that work in mixes (e.g., delay); those are called high-latency anonymous communication systems for a reason!

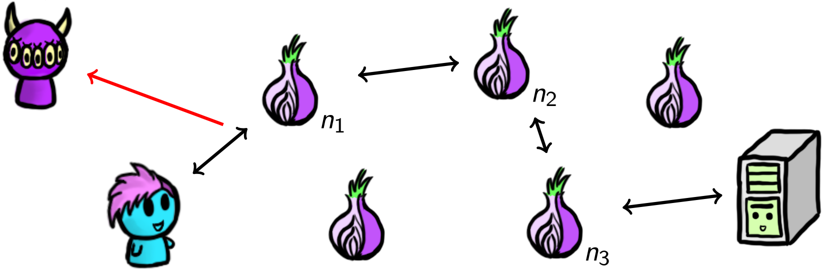
Tor and Internet censorship

- State-level adversaries can restrict connections to public Tor relays or otherwise attempt to fingerprint Tor traffic on the network
- Solution?
  - Distribute addresses of non-public Tor relays (bridges)
  - Modify Tor traffic to look like something else (pluggable transports)



[1] Picture from Matic et. al, NDSS'17 paper

Other threats: website fingerprinting over Tor



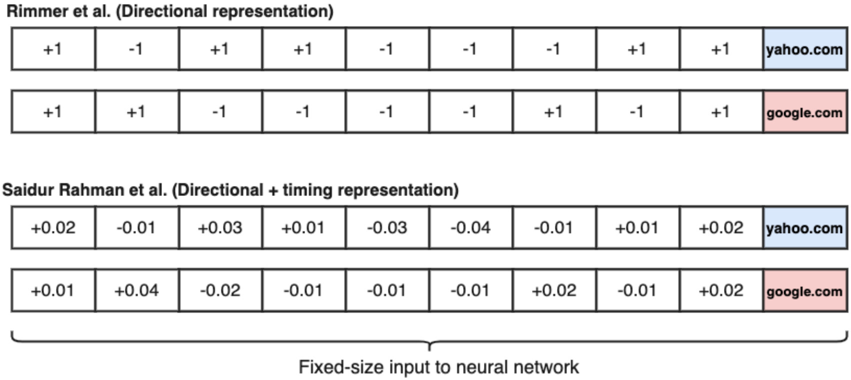
- Eve can passively observe traffic between Alice and the first node
- This traffic is encrypted... but does it reveal any information about Alice?

Website fingerprinting

- An encrypted connection still leaks:
  - Source and destination IP addresses.
  - Source and destination ports.
  - Flow duration.
  - Amount of packets exchanged.
  - Packet sizes.
  - Inter-arrival times.
  - . . .
- Without Tor:
  - Eve can gather a training set that contains these features associated to different websites.
  - When Alice connects to a website (e.g., using TLS), Eve can run a classifier to decide which of the websites Alice was connecting to.
- With Tor: even though Tor exchanges data in fixed-size cells, packet direction and timing still leaks information

Website fingerprinting over Tor

- Learned features based on different traffic representations can be used to launch website fingerprinting attacks on Tor
  - Directional representation Rimmer et al., NDSS '18
  - Directional + timing representation Saidur Rahman et al., PoPETs '20



Rimmer et al. (Directional representation)

+1	-1	+1	+1	-1	-1	-1	+1	+1	yahoo.com
+1	+1	-1	-1	-1	-1	+1	-1	+1	google.com

Saidur Rahman et al. (Directional + timing representation)

+0.02	-0.01	+0.03	+0.01	-0.03	-0.04	-0.01	+0.01	+0.02	yahoo.com
+0.01	+0.04	-0.02	-0.01	-0.01	-0.01	+0.02	-0.01	+0.02	google.com

Fixed-size input to neural network

Anonymity vs. pseudonymity in Tor

Tor provides for anonymity in TCP connections over the Internet, both unlinkably (long-term) and linkably (short-term)

What does this mean?

- There's no long-term identifier for a Tor user
- If a web server gets a connection from Tor today, and another one tomorrow, it won't be able to tell whether those are from the same person
- But two connections in quick succession from the same Tor node are more likely to come from the same person

Module outline

16 Privacy and Anonymity

17 Mixes

18 Attacks on Mixes

19 A Brief History of Remailers

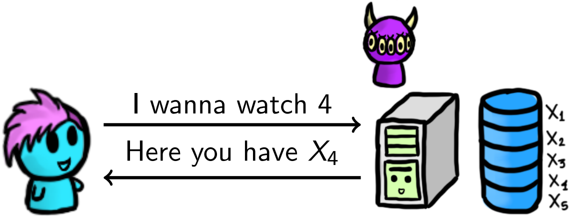
20 Tor

21 Private information retrieval (PIR)

Motivation

Simple scenario:

- Netflix stores it's movies in a database
    1. Legally Blonde
    2. The Godfather
    3. The Dark Knight
    4. Mean Girls
    5. ...



I wanna watch 4

Here you have X_4

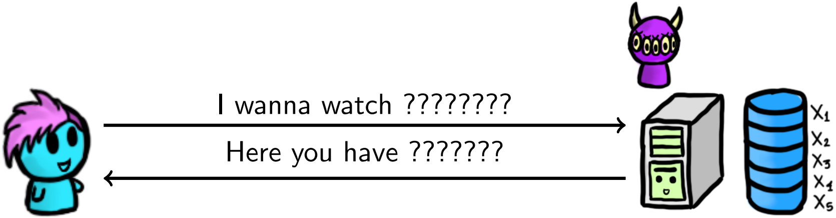
- You request movies by index
- Netflix caches your selection and gradually builds a profile on your movie preferences
- But why? You have paid for a Netflix license and so you should be able to access different movies

Definition

Goal: allow a user to query a database while hiding the identity of the data items the user is after

Formal model:

- Server: holds an n-bit string $\{X_1, X_2, \ldots, X_n\}$
- User: wishes to retrieve X_i AND keep i private



I wanna watch ????????

Here you have ???????

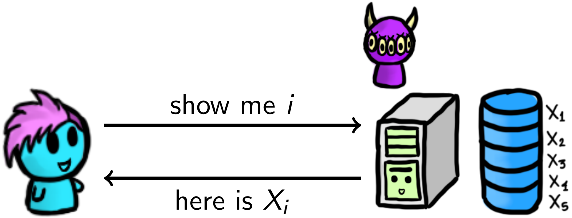
Non-private protocol

Formal model:

- Server: holds an n-bit string $\{X_1, X_2, \ldots, X_n\}$
- User: wishes to retrieve X_i AND keep i private

Protocol:

- User: show me i
- Server: here is X_i



show me i

here is X_i

Q: Privacy and Bandwidth?

A: No privacy! But very efficient, since we just receive 1 file (in this example, 1 bit)

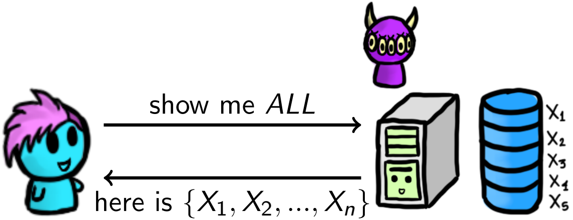
Trivially-private protocol

Formal model:

- Server: holds an n-bit string $\{X_1, X_2, ..., X_n\}$
- User: wishes to retrieve X_i AND keep i private

Protocol:

- User: show me *ALL*
- Server: here is $\{X_1, X_2, ..., X_n\}$



Q: Privacy and Bandwidth?

A: Total privacy! But we receive n files (in this example, n bits)

Sad news: if the server has unlimited computational power AND there is only a single copy of the database $\Longrightarrow$ n bits must be transferred!

“More” solutions?

- User asks for additional random indices
  - Drawback: balance information leak vs communication cost
- Note: anonymity is a different concern: it hides the identity of the user, not the fact that X_i is retrieved.
- We will see two PIR solutions:
  - Information-theoretic PIR (IT-PIR)
  - Computational PIR (CPIR)

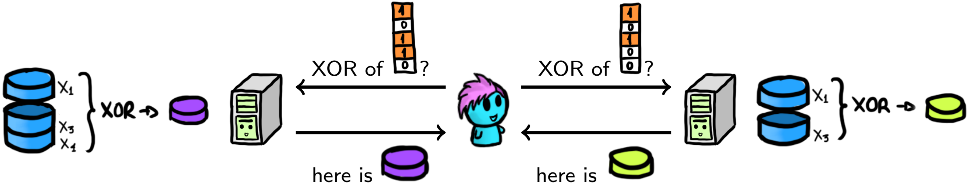
Information-theoretic PIR (IT-PIR): visually

- There are two non-colluding servers with a copy of the dataset

1. Alice wants X_4: she generates a random n-length binary vector and XOR's it with e_4 (where e_4 is an all-zero vector with it's $4th$ entry set to 1).

1 0 1 1 0
1 0 1 0 0 = 1 0 1 1 0 ⊕ 0 0 0 1 0

2. Alice queries the servers for XORs of entries:

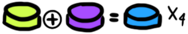


XOR of ?

here is

XOR of ?

here is

3. Alice recovers the desired element: 

Information-theoretic PIR (IT-PIR): formally

Formal model:

- Server: holds an n-bit string $\{X_1, X_2, ..., X_n\}$
- User: wishes to retrieve X_i AND keep i private

Assumption: multiple (≥ 2) non-cooperating servers

An example 2-server IT-PIR protocol:

- User → Server 1: $Q_1 \subset_R \{1, 2, ..., n\}$ Choose each index at random with prob 50%
- Server 1 → User: $R_1 = \bigoplus_{k \in Q_1} X_k$
- User → Server 2: $Q_2 = Q_1 \Delta \{i\}$ Add i if it was not in Q_1, otherwise remove it
- Server 2 → User: $R_2 = \bigoplus_{k \in Q_2} X_k$
- User derives $X_i = R_1 \oplus R_2$

Q: Privacy and Bandwidth?

A: Total privacy! (if servers do not collude). We receive 2 files (2 bits), and this requires some inexpensive computation

Some notes about IT-PIR

Formal model:

- Server: holds an n-bit string $\{X_1, X_2, ..., X_n\}$
- User: wishes to retrieve X_i AND keep i private

Assumption: multiple (≥ 2) non-cooperating servers

An example 2-server IT-PIR protocol:

- User → Server 1: $Q_1 \subset_R \{1, 2, ..., n\}$ Choose each index at random with prob 50%
- Server 1 → User: $R_1 = \bigoplus_{k \in Q_1} X_k$
- User → Server 2: $Q_2 = Q_1 \Delta \{i\}$ Add i if it was not in Q_1, otherwise remove it
- Server 2 → User: $R_2 = \bigoplus_{k \in Q_2} X_k$
- User derives $X_i = R_1 \oplus R_2$

In this example, we also send $2 \cdot n$ bits (Q_1 and Q_2) to the servers; there is a better way of doing this that requires $O(\log n)$ bits.

Practice: IT-PIR

Two non-colluding servers (S_1 and S_2) have an identical copy of a dataset that contains six 5-bit files (indexed by $i = 1, 2, \ldots, 6$). The dataset is given by the following matrix (each row is a file):

$$\mathbf{D} = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 \end{bmatrix}$$

Alice wants to retrieve the $3rd$ file, but doesn't want to reveal this information to the servers. She generates a random binary vector $Q_1 = [0, 1, 1, 0, 1, 0]$ and sends it to S_1.

Q: What does she receive from S_1? What does she send to S_2? What does she receive from S_2? What is the XOR of the two received values? (it should be X_3)

A: [1,0,0,1,1], [0,1,0,0,1,0], [1,0,1,1,1], [0,0,1,0,0]

Computational PIR

Formal model:

- Server: holds an n-bit string $\{X_1, X_2, \ldots, X_n\}$
- User: wishes to retrieve X_i AND keep i private

Quadratic residue

a is QR mod m if

$$z^2 \equiv a \pmod{m}$$

Assumption: A single server with limited computational power

An example CPIR protocol:

- User chooses a large random number m
- User generates $n-1$ random quadratic residues (QRs) mod m: $a_1, a_2, \ldots, a_{i-1}, a_{i+1}, \ldots, a_n$
- User generates a quadratic non-residue (QNR) mod m: b_i
- User → Server: $a_1, a_2, \ldots, a_{i-1}, b_i, a_{i+1}, \ldots, a_n$
- Server cannot distinguish between QRs and QNRs mod m, i.e., the request is just a series of random numbers: $u_1, u_2, \ldots, u_n$
- Server → User: $R = u_1^{X_1} \cdot u_2^{X_2} \cdot \ldots \cdot u_n^{X_n}$
- If R is a QR mod m, $X_i = 0$, else (R is a QNR mod m) $X_i = 1$

Comparison of CPIR and IT-PIR

CPIR

- Possible with a single server
- Server needs to perform intensive computations
- To break it, the server needs to solve a hard problem

IT-PIR

- Only possible with > 1 server.
- Server may need lightweight computations only
- To break it, the server needs to collude with other servers

Big M5 recap (non-exhaustive list of things you need to know)

First part: cryptography

- Basic definitions: CIA in crypto, Kerckhoff's principle, trying every key (times), passive vs active adversary, secret-key vs public-key properties, hybrid encryption, repudiation vs non-repudiation, etc.
- Two-time pad questions (you have a question like this in quiz 5), also block cipher mode of operation questions (also on quiz 5)
- Malleable encryption questions (we saw textbook RSA, you have it also in A3 written)
- We give you keys available to Alice and Bob and ask you to write things like:
    - MAC-then-encrypt, Encrypt-then-MAC, combined with hybrid encryption, signatures (hash-then-sign)
    - We could give you an approach Alice and Bob follow and ask if Mallory could do MITM, and then ask you to write the formulas of how Mallory would proceed
    - ...
- (Basically, you need to *understand* why and how we use these crypto tools)
- Diffie-Hellman, remember how it works, maybe we can ask you to write it with numbers, etc.
- The key management problem (with manual keying like SSH, web of trust like PGP, or certificate authorities like TLS)

Big M5 recap (non-exhaustive list of things you need to know)

Second part: cryptography applications

- Crypto in program and OS security, encrypted code, encrypted drives
- WEP: recognize similar issues in other protocols, or describe these issues and how to fix them
- IPSec: two modes and two headers, what is encrypted/authenticated on each?
- TLS: how CAs work, scenarios with Mallory trying to MITM Alice and the web server
- SSH: TOFU, authenticating the server and the client
- PGP: web-of-trust (signing other people's public keys with our private keys, so that our friends with our public keys trust what we sign)
- OTR: recognize whether protocols give forward secrecy or post-compromise security, some questions about the OTR ratchet

Big M5 recap (non-exhaustive list of things you need to know)

Third part: PETs

- Understand differences between data and meta-data, and what anonymity means.
- Two big classes of anonymous communication systems, typically called high-latency (mixes) and low-latency (Tor). Understand the pros and cons of each.
- Mixes:
  - Three basic operations (change appearance, delay, dummy/cover traffic), there are many types according to how they add delay (threshold, timed, pool, continuous-time), and many topologies (cascade, free-route, stratified). Understand pros and cons of each mix and topology.
  - Long-term profiling attacks: understand why these work, and know how to compute an SDA profile estimation.
- Tor:
  - Understand how building a circuit works.
  - What can a global passive adversary learn? (website fingerprinting, traffic correlation)
  - Role of bridges in bypassing censorship
- PIR: there are two types, CPIR and IT-PIR. Understand pros and cons of each.
  - IT-PIR: Get familiar with how and why IT-PIR works. Understand why it provides privacy. Know how to compute the bandwidth cost (maybe we could explain a variant and ask things about it)
  - CPIR: understand the example CPIR protocol, and the privacy guarantees it provides (maybe we could explain a variant and ask things about it)