Computer Networking and IT Security (CNS)

INHN0012 – WiSe 2025/26

Lorenz Lehle, Stephan Günther

Chair of Distributed Systems and Security
School of Computation, Information and Technology
Technical University of Munich

Wednesday 28th January, 2026 11:36

Chapter 6: IT Security

Chapter 6: IT Security

Chapter 6: IT Security

Basics

Terminology

Security Goals

- Data Integrity
  - No improper or unauthorized change of data
- Confidentiality
  - Concealment of information
- Availability
  - Services should be available and function correctly
- Authenticity
  - An entity is who it claims to be
- Accountability
  - Identify the entity responsible for any event
- Controlled Access
  - Only authorized entities can access certain services or information

Basics

Terminology

Authentication vs. Authorization

- Authentication
    - Proves an entity is who it claims to be
    - Security goal: Authenticity
- Authorization
    - Specifies which rights an entity has on a given set of resources
    - Security goal: Controlled Access

Basics

Terminology

Vulnerability

- A flaw in a computer system that weakens its security
  - Buffer overflows
  - Input sanitization
  - The user
  - . . .

Attack Vector

- A way of chaining one or more vulnerabilities
- An attack vector is used to achieve the attackers goal
  - Access to the system
  - Information gain
  - . . .

Exploit

- A piece of software or a sequence of commands
- Leads to (programmer) unintended behavior on the target (program)
- The attacker uses exploits to leverage attack vectors

Basics

Terminology

Security Policy

- A security policy defines the desired state of a system regarding its security
  - e.g. “Confidential information may only be read by parties of a certain privilege level” or
  - e.g. “Traffic addressed to this IP must be encrypted before being sent”
- A security policy is realized by security mechanisms, such as firewalls and access control systems

Risk

- A vulnerability is exploited with probability P
- The damage induced is estimated with value E
- The risk is $R = P \cdot E$
- If $R <$ *cost of fix*, a company is likely not interested in fixing

Privilege Escalation

- Artificially increasing the privilege of the (harmful) program on the host
- May use operating system vulnerabilities to achieve this

Basics

Terminology

Malware

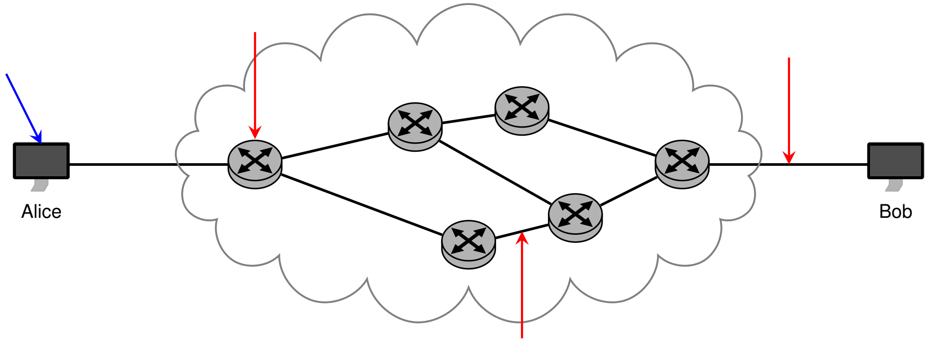
- An attacker crafted piece of software designed to achieve attacker defined goals
  - Deployed on the attacked system
  - $\rightarrow$ Goals: Steal information, disrupt services, enable access, . . .

Types of Malware

- Virus: harmful program, does not self-replicate.
- Worm: harmful program, self-replicating. Potentially autonomous.
- Spyware, Adware: Try to steal information (e.g. credit cards) from the user, the classic Bing toolbar
- Rootkits: Enables privilege escalation on the host. Hidden in the operating system (kernel).
- Ransomware: Encrypts data and holds the user for ransom. Optional: extortion schemes

Basics

Attackers in the Network



IT Security, Network Security and Host Security

- IT Security is the general field that also contains host and network security
- Host security is mostly concerned with malware, input sanitization, SQL injections, . . .
- Network security describes the security of the communication protocols used

Basics

Defining the Attacker

Passive Attackers

- Passively listen to network traffic
- Try to gain information through analysis
- May leverage protocol vulnerabilities to do so

Active Attackers

- Additionally actively tamper with traffic
- Leveraging some vulnerabilities requires an attacker to become active
  - Replaying packets
  - Changing sequence numbers
  - ...
- A attacker who controls all traffic between two clients is called man-in-the-middle (MitM)

Types of Attackers

- Script Kiddies, Insiders, Hacktivists, Cybercriminals, Nation-State

! You have to decide what you want to defend against, and chose appropriate measures

Basics

A Collection of Threats

- Masquerade
  - An entity claims the identity of another entity and acts as it
- Eavesdropping
  - An attacker passively reads traffic
- Ransomware, double extortion
  - User data is encrypted and a ransom demanded for decryption
  - A double extortion scheme bypasses backups by stealing and threatening to leak information
- (Distributed) Denial of Service (D)DoS
  - The availability of the service or network is affected

  → Overloading a service

  → Jamming the Wi-Fi frequency
- Authorization violation
  - An attacker accesses resources they should not be able to access

Chapter 6: IT Security

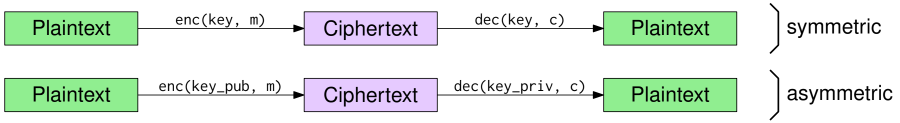
Encryption

A High Level View

The Purpose of Encryption

- The purpose of encryption is to hide information
  - We want to prevent an attacker from deriving any information from encrypted data
  - We do not only want to protect the data, but also hide its properties
- The ciphertext shall only be useful to such parties that hold the secret
- A perfect encryption scheme makes the result indistinguishable from randomness

- We distinguish two types of encryption:



Encryption

The Key(s) to the Castle

! We differentiate two types of encryption, based on the types of key used

Symmetric Cryptography

- Symmetric encryption uses the same key for encryption and decryption
- Therefore, the key has to be exchanged in advance
- Symmetric encryption schemes usually are very fast
- Examples: AES, RC4, ChaCha20

Asymmetric Cryptography (aka Public-Key Cryptography)

- Asymmetric encryption is distinguished by a key pair
  - The private key is only known to the receiver
  - The public key is known to everybody
- Data is encrypted using the public key
  - Once encrypted, the ciphertext can only be decrypted using the private key
- Therefore, the (secret!) keys do not have to be exchanged in advance
- Asymmetric schemes in general are slow
- Examples: RSA, ECC

Hybrid Encryption Schemes

- Often, a symmetric key is established using a asymmetric scheme
- This key is then used for symmetric encryption, making use of its throughput advantage

Encryption

Hiding Patterns

- Making (just the) information inaccessible is easy
- Hiding the patterns is way harder
  - An attacker might still be able to extract information from the patterns alone
- Just using encryption will not save you
  - ! You also have to use it correctly

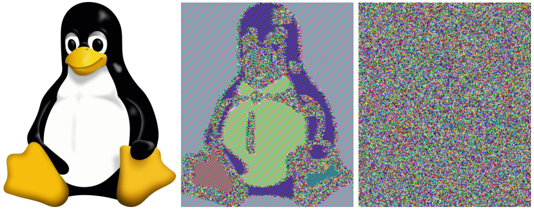


Figure 1: The same image encrypted using AES-ECB (middle) and a different AES mode (right). Source: Wikipedia[1]

- We strive to make the ciphertext indistinguishable from random noise
  - ⇒ If the data has true random properties, the attacker cannot learn anything from it

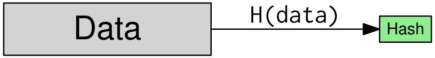
[1] `https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation`, CC0 and CC BY-SA 4.0

Chapter 6: IT Security

Authentication and Trust

Hash Functions

- Before we can continue to authentication, we have to introduce hash functions



- A hash function reduces an arbitrary amount of data to a deterministic characteristic value (e.g. 256 bit)

- We want the following to hold:
  - Two identical inputs result in the same hash (**deterministic**)
  - Changing the input only little results in a very different hash (**unpredictable**)
  - It should be hard to find the input for a given hash (**one-way property**)
  - It should be hard to find a second input that results in the same hash (**collision resistance**)

- We use above conditions to define cryptographic hash functions

Authentication and Trust

Cryptographic Hash Functions

A cryptographic hash function fulfills three properties:

1. **Pre-Image Resistance:** Given a hash value, it is hard to find an input that results in the same hash.
2. **Second Pre-Image Resistance:** Given a message, it is difficult to find another input that results in the same hash.
3. **Collision Resistance:** It is difficult to find a pair of two different messages that result in the same hash.

Examples of cryptographic hash functions:

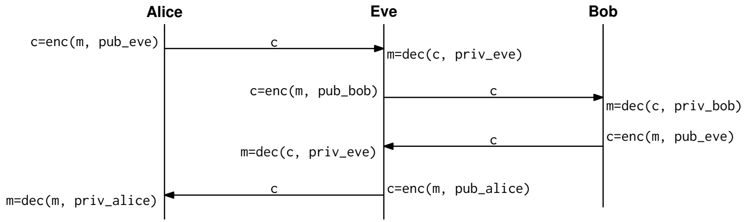
- MD5 (horribly broken)
- SHA-1 (broken)
- SHA-2, SHA-3
- BLAKE2, BLAKE3
- KangarooTwelve
- . . .

Authentication and Trust

Why do we need Authenticity?

- When connecting to a host in the internet, you want to ensure that the host is who they claim to be
- Otherwise, you are vulnerable to man-in-the-middle attacks

- Assume: A simple asymmetrically encrypted communication, and eve has managed to
  - tell Alice that Bob's public key is `pub_eve`
  - tell Bob that Alice's public key is `pub_eve`



- Eve can now MitM all communication without either alice or bob noticing
- We therefore need mechanisms to enable data origin authenticity

  ⇒ This would allow Bob and Alice to detect the MitM

Authentication and Trust

Providing Authenticity

- Asymmetric schemes inherently provide no sender authenticity
  - By definition, everybody can encrypt data using your **public** key
- Symmetric schemes are better in this regard
  - They implicitly provide authentication since the key is only known to the participating parties
  - Holding a syntactically correct result after decryption provides at least some sender authenticity
  - Somebody who does not know the key cannot forge a ciphertext for a given plaintext
  - Still, we cannot be completely sure of the sender's identity

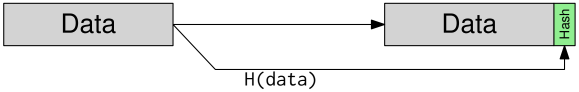
  $\rightarrow$ The ciphertext could have been replayed

- We therefore want to ensure:
  1. Data authenticity: We can verify that the data has been sent by the entity that claims to have sent it
  2. Data integrity: If the data has been tampered with, we can detect it

Authentication and Trust

Providing Authenticity

- One could try to provide data authenticity and data integrity by including a hash of the message
  - ! While this seems like a good idea, it does not work (at all)



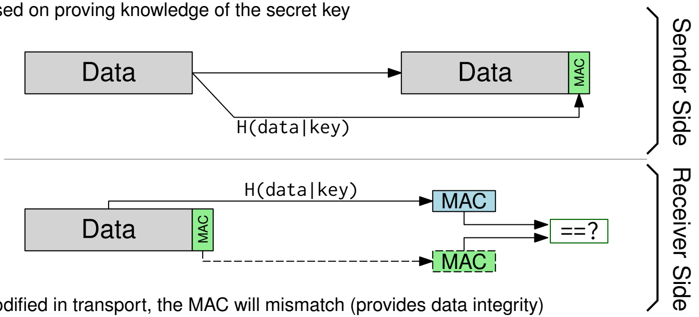
- Cryptographic hashes do **not** protect against tampering
  - ! This is, as an attacker can just recalculate the hash (the hash function is not secret)
  - They neither provide data authenticity nor data integrity
  - If the hash is communicated in a secure channel, a cryptographic hash function can provide data integrity

- Cryptographic hash function still can be extended to provide authentication
  - → We shall continue with defining a simple authentication scheme using a cryptographic hash function

Authentication and Trust

Data Authentication: A Simple Scheme

- We introduce the concept of a Message Authentication Code (MAC)
- Both sides share a common secret key. They include it in the MAC
  - → This MAC is sent along the message
- Authentication is based on proving knowledge of the secret key



- If the message is modified in transport, the MAC will mismatch (provides data integrity)
- An attacker cannot forge MACs as they do not know the secret
- The schemes used in the real world (e.g. HMAC) are a bit more complex, but based on the same fundamentals

Question: Do MACs protect against replays? If not, how can we extend the scheme so that it does?

Authentication and Trust

A Matter of Trust

- Establishing authenticity relies on a long-term secret (“the key”) exchanged in advance through a trusted channel
  - The secret can be exchanged in advance
  - Alternatively, a trusted third party can vouch for the secret, and thereby the identity of the owner
  - Latter scheme is what you will find in real world usage most of the time
  - That is, because it is infeasible to exchange secrets with everybody in the internet in advance

Trusting the Scheme

- Classic authenticity schemes work with proving knowledge of a secret only known to both sides
- MACs and symmetric cryptography inherently include this secret, and thereby provide authenticity
- Asymmetric encryption has no such pre-established secret to rely on

Who can you trust?

? So how can we build trust in a network so large that it is not possible to share (pairwise) secrets in advance?

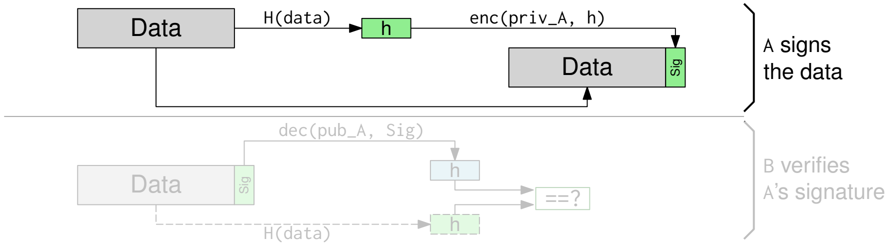
  $\rightarrow$ Exchanging pairwise secrets in advance does not scale well: n nodes $\Rightarrow \frac{(n-1)n}{2}$ keys

Authentication and Trust

Digital Signatures: Signing

- Assume, an entity wants to prove its identity without having to establish a common secret
- In asymmetric cryptography, we can also “reverse” the encryption process
  - We encrypt using the private key, and anybody possessing the public key can decrypt
  - This is, by definition, everybody
- The private key `priv_A` is only known to `A` itself, therefore, only `A` can use it
- Assume, `A` wants to sign a given piece of data. They . . .
  1. hash the data using a cryptographic hash function
  2. encrypt the hash using its private key (only `A` can do this!)
  3. send the data alongside the hash

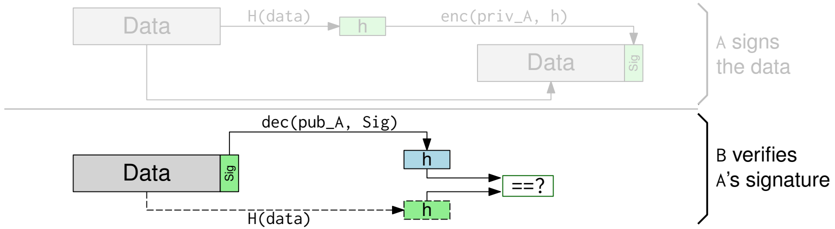
  → Assuming we can trust the relation `priv_A↔A`, this can be extended to an authentication scheme



Authentication and Trust

Digital Signatures: Verifying

- Anybody who knows A's public key (everybody) can now check the signature:
  1. Hash the data
  2. Decrypt the signature using A's public key
  3. Check whether the hashes are equal



- Since only A knows `priv_A`, only A can generate valid signatures
- If an attacker changes the data, the hash will change, and therefore the signature will mismatch

! One problem is still unsolved: How do we prove the relation `priv_A`↔A?
→ That is: how can we trust, that `priv_A` is the private key of A and not Eve?

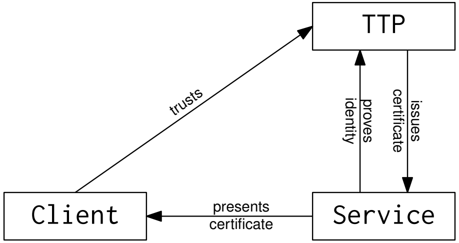
Chapter 6: IT Security

Trusted Third Parties and Public Key Infrastructures

The Principle of a Trusted Third Party

- It is infeasible to exchange secrets in advance with every entity on the internet
- But what, if we trusted a handful of trust providers:
  - A service A (or rather its operator) proves its identity to this party P
  - We trust P to correctly check the identity
  - P hands A a certificate of its identity
  - We can use this certificate to verify A’s identity
- We call P a Trusted Third Party (TTP)

- Now, when communicating with A, they will hand us their certificate
  - This certificate proves to us that P has validated A’s identity
- As we trust P, we can trust the certificate of A and thereby its identity



Trusted Third Parties and Public Key Infrastructures

Certificates and Certificate Authorities

- So how can a TTP prove that it checked an entities identity

→ Introducing: The concept of certificates

! The real implementations are a bit more complex, we shall stick to the concepts for this chapter

Certificate Authorities

- A TTP whose purpose is to verify and sign the identity of others is called a Certificate Authority (CA)
- A CA is comparable to e.g. the state that provides you with an identification document
  - The main difference is that what a CA does is digital — and therefore there are no shiny marks on the document that can prove its authenticity

  ⇒ We need a digital counterpart for such shiny marks

Certificates

- A certificate contains, similar to real life, information about an entity (such as DNS names, or your name within an organization)
- The widely adopted standard is X.509
- A certificate actually is a key pair
  - The public key is contained in the certificate itself
  - The private key stays with the entity, it uses it to prove that it is owner of the certificate
- At the end of the certificate we find a signature
  - This signature proves, that the information of the certificate was checked by a TTP
  - It is a hash of the certificate that is then encrypted using the private key of the TTP
- Additionally, certificates are only valid for a given time span (e.g. one year)

Trusted Third Parties and Public Key Infrastructures

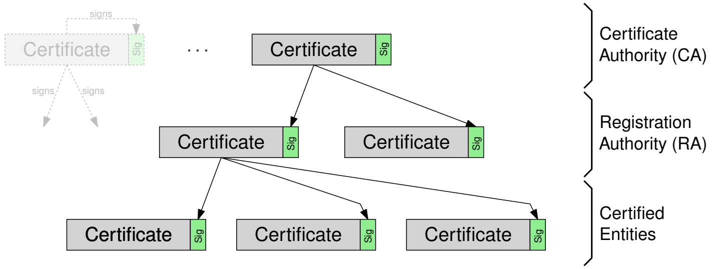
An Example

```
$ openssl x509 -noout -text -in example.com.crt
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 18031233244003276 50
        Signature Algorithm: sha384WithRSAEncryption
        issuer: C = US, O = Let's Encrypt, CN = R3
        Validity
            Not Before: Dec 21 21:58:58 2021 GMT
            Not After : Dec 20 21:58:58 2024 GMT
        Subject: CN = example.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (4096 bit)
                Modulus:
                [...]
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Authority Key Identifier:
                keyid:FA:D1:C7:0D:86:89:AF:93:CF:F6:33:25:12:C5:75:99:76:52:25:AC
            X509v3 Subject Alternative Name:
                DNS:example.com
            X509v3 Extended Key Usage:
                TLS Web Client Authentication
    Signature Algorithm: sha384WithRSAEncryption
    11:24:b3:9c:85:99:04:42:fc:1f:b9:87:c9:c7:db:89:41:f3:
    [...]
```

Trusted Third Parties and Public Key Infrastructures

A Hierarchy of Trust

- So our hierarchy of trust looks somewhat like this:



- We can trust any level that has been signed from above, as long as we trust the root
- We put large values of trust in this root, as it can sign arbitrary identities and we will trust them
- There exist a collection of roots worldwide, whose certificates are shipped with your browser or OS

Trusted Third Parties and Public Key Infrastructures

A Word on Hierarchies

CA vs. RA

- For the internet, the CA does not take care of verifying each requesters identity
  - Registration Authorities (RA) take care of this
  - They check the identity (possibly according to local law)
  - Once completed, they ask the CA to issue a certificate to the entity
  - RAs do not issue certificates!

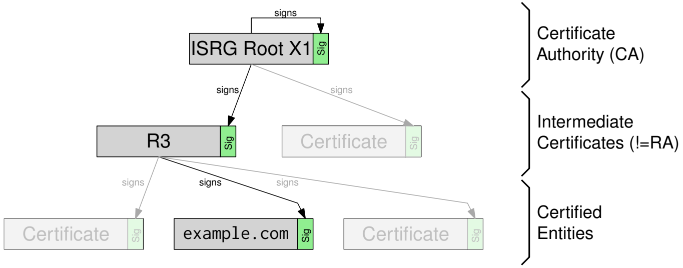
Two different hierarchies

- Note that we look at two different hierarchies when talking about PKIs
  1. The scheme of trust, which spans from CA to RA and the entity
  2. The actual hierarchy of certificates, from root, over intermediate certificates to the entity
- The intermediate certificates are used by CAs to be able to rotate certificates
  - This prevents having to change the root certificates (which involves you updating your browser/OS)
  - Rotating an intermediate certificate would e.g. become necessary when it expires

Trusted Third Parties and Public Key Infrastructures

Verifying an Identity

- Assume service X wants to prove to you that they really are X
- X has a certificate by a CA you trust (indirectly, usually there are multiple levels)
- X proves to you that they are owner of their certificate by using the associated private key to encrypt a challenge you sent
  - Only X can do that: if you can decrypt using the public key from the certificate, X indeed is the owner of the certificate



- You now try to establish a chain of signatures from one of the roots down to X's certificate
  - That is, check if you can decrypt the signature using the public key of the entity who has signed
  - If you are able to decrypt, that entity actually signed, as only they have the associated private key
  - If you can repeat this process until you reach a trusted root, you can trust X's certificate and thereby their identity

Chapter 6: IT Security

Random

Why not all Randomness is True

- We have seen the use of secrets throughout this chapter
- To achieve the properties we wish for, we need these secrets to be absolutely unpredictable
  - ! If an attacker can predict what secrets we generate, they can break e.g. the encryption based on it
- True randomness is hard to achieve for computers
  - Computers are good at being predictable
  - They are not especially good at behaving unpredictably
- Therefore, when generating randomness for key generation, we fall back to external inputs
  - Factors such as mouse movements and network traffic properties are fairly unpredictable
  - Modern CPUs also feature an HRNG (hardware random number generator)
  - → An HRNG is a circuit that is specially designed to behave instable, and therefore random
- GNU/Linux offers two random sources
  - `/dev/random`: Random data for cryptographic purposes, sustains a minimum level of entropy
    - → Will block if not enough random input (seed material) is available
  - `/dev/urandom`: Lower quality random data, can fall under the minimum level of entropy
    - → Never runs out, and therefore does not block

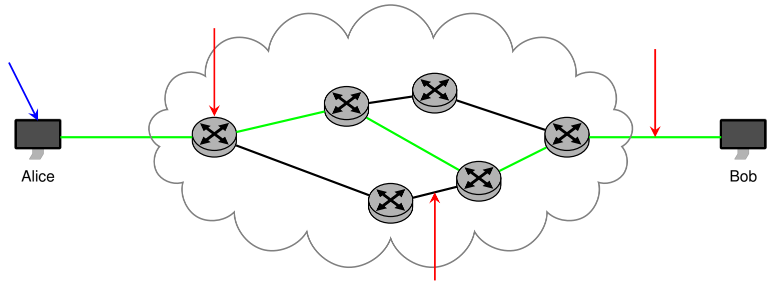
⇒ The main takeaway should be: don't use `init(get_time_ms())` to initialize your random generator

Chapter 6: IT Security

Secure Channels: IPsec

The Internet is a Dangerous Place

- Your connection to a server passes many nodes, none of which you control
- It is therefore best to assume them untrusted, and build your protocols to withstand this



- We are interested in establishing a trusted communication channel through an untrusted network
  - → Such channel is called a secure channel
- Secure channels can be established on different layers
  - → We will look at IPsec (layer 3) in the following
  - ! Forget about application layer protocols (e.g. TLS) that guarantee security for a second

Secure Channels: IPsec

What IPsec Tries to Solve

IPv4 and IPv6 cannot guarantee:

1. Data origin authentication
   - Whether the source address of the packet is actually the source
2. Data integrity
   - Whether the payload has been changed in transit
3. Confidentiality
   - Third parties can inspect the packet payload

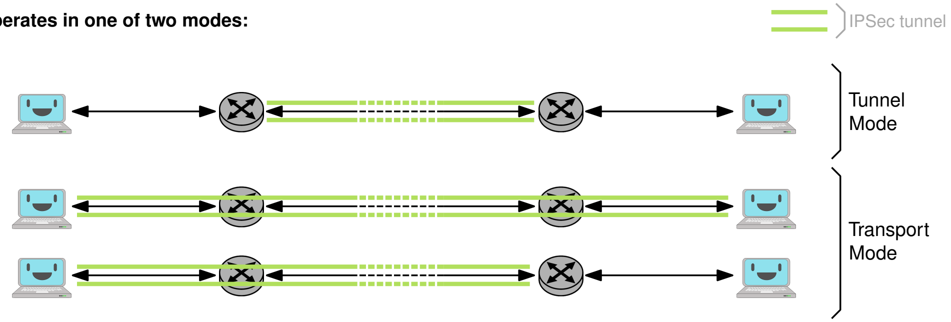
IPsec tries to solve this. It allows implementing a security policy for traffic.

- IPsec can be used to implement a Virtual Private Network (VPN)
- Still, IPsec in itself is only a means to secure layer 3 payloads in transit
- IPsec is complex to configure, but it is a very mighty tool
  - It allows for security seamlessly integrated in the network stack
  - It is policy based, being a single policy decision point in the stack
  - This differs from typical VPN approaches such as OpenVPN and Wireguard

Secure Channels: IPsec

Tunnel and Transport Mode

IPsec operates in one of two modes:



IPsec can secure parts of the path, or the entire path

- In **tunnel mode**, traffic is encrypted on a path between two routers
  - ⇒ The existing IP packet is seen as payload of a new IP packet addressed to the other tunnel end
- In **transport mode**, traffic is encrypted between the end hosts
  - Since the ends of the tunnel are equal to the ends of the IP connection, no new header has to be added
  - The original IP header is used for addressing the ends of the tunnel

Secure Channels: IPsec

With great Power comes great Responsibility

IPsec enables complex traffic processing. An implementation has to provide:

- Securing traffic, providing confidentiality, authenticity and integrity
- Matching traffic against a policy and decide accordingly
- Establish and handle key material for multiple tunnels on the fly
- Being robust against active attackers injecting and modifying traffic
- Handling packet loss and thereby data missing during decryption
- Handling packets arriving out of order and still decrypting such traffic

If IPsec is correctly deployed it provides:

1. Data origin authentication
2. Connection-less data integrity
3. Confidentiality
4. Security Policies
   → Nodes can determine the minimum security level for traffic and drop it if it doesn't meet them

Secure Channels: IPsec

Establishing the Tunnel

A Role Model

- IPsec nodes have one of two roles: initiator and responder
- The initiator contacts the responder to establish a tunnel
- The roles are not necessarily fixed
  - Commonly, only endpoints with a static address are responders

Establishing Keys: IKE

- IPsec establishes the encryption and authentication/integrity keys via IKE/IKEv2
- The Internet Key Exchange (IKE) protocol is used to establish the tunnel and its keys
- It handles:
  1. Authentication of the endpoints
  2. Cipher suite negotiation
  3. Key establishment
  4. IPsec related control tasks (such as setting up and tearing down tunnels)

- IKE/IKEv2 are complex protocols, their details are therefore not in the scope of this lecture

Secure Channels: IPsec

AH and ESP

Each IPsec tunnel uses one or both of the following protocols:

AH (Authentication Header)

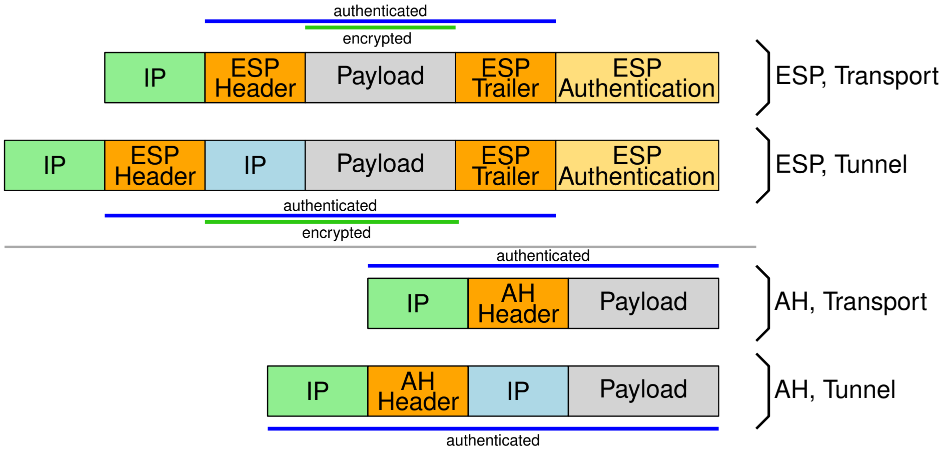
- Provides data authentication and data integrity for the payload and parts of the preceding IP header
- ESP is considered superior since computation capacity is a non-issue

ESP (Encapsulating Security Payload)

- Provides Data Authentication, Data Integrity and Data Confidentiality
- Does **not** protect the preceding IP header
    - This is a non-issue in reality as authenticity can be provided by the application layer

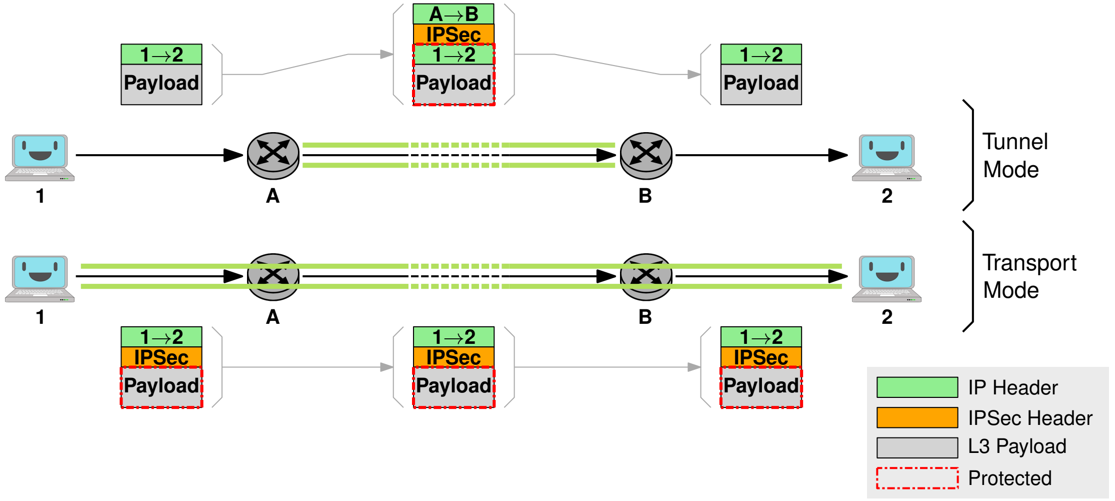
Secure Channels: IPsec

AH and ESP Headers and Protection



Secure Channels: IPsec

Protection in transit



Secure Channels: IPsec

Securing Traffic

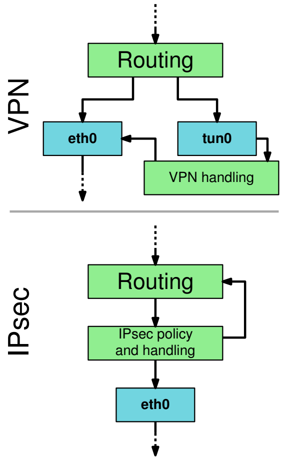
IPsec integrates into layer 3 of the network stack.

- A ruleset defines whether and how to handle traffic
  - We match flows of traffic and decide how to handle them
  - By handling flows based on a ruleset a policy is enforced
    - e. g. "Secure all traffic going from site1 to site2"
- Rules follow the syntax `(src ip, dest ip, protocol)`$\longrightarrow$`(none/discard/protect)`
  - e. g. `(192.168.60.0/24, 10.24.0.0/16, all)`$\longrightarrow$`(protect)`
  - e. g. `(192.168.60.0/24, 192.168.40.0/24, all)`$\longrightarrow$`(none)`
- The rulesets on both ends of the tunnel must reflect each other
- If a packet matches a rule, it is handled accordingly
  - IPsec creates the tunnel on-demand, as soon as the first packet matches the according policy
- If a packet matches the reverse of a rule on a host, the IPsec handling is reversed (e.g. decryption of payload)
- After handling, the result is injected back into the top of layer 3

Secure Channels: IPsec

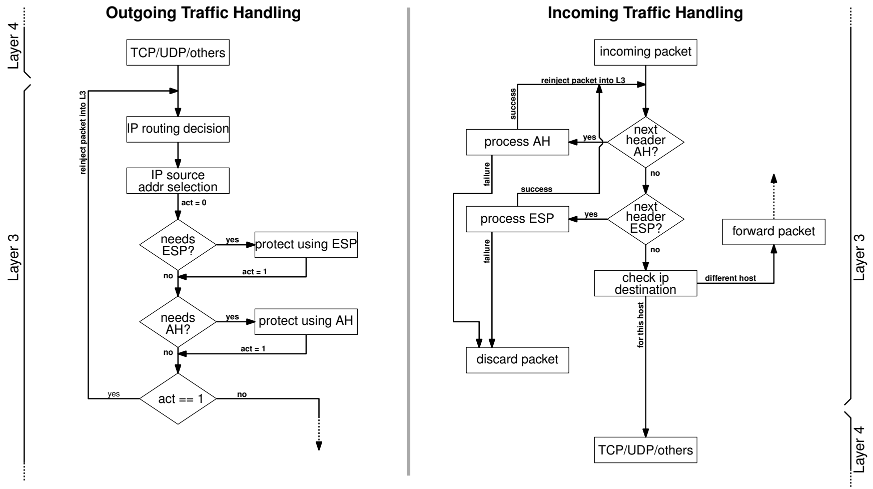
Integrating into the Network Stack

- IPsec can be used to construct VPNs, but it is mightier than a VPN
- It integrates differently into the network stack compared to more traditional VPN approaches such as OpenVPN or Wireguard
- Typical VPN approaches install a virtual network interface
  - Traffic routed through this interface gets tunneled to the other side of the VPN tunnel
  - It is protected in transit by the VPN protocol
  - The policy is therefore reflected in the routing table
- In contrast, IPsec is policy based
- It inspects all traffic passing through L3 of the network stack
  - If a packet matches a policy, it is handled accordingly and re-injected in the network stack
- Thereby IPsec can cope without virtual network interfaces



Secure Channels: IPsec

IPsec Traffic Handling [1]

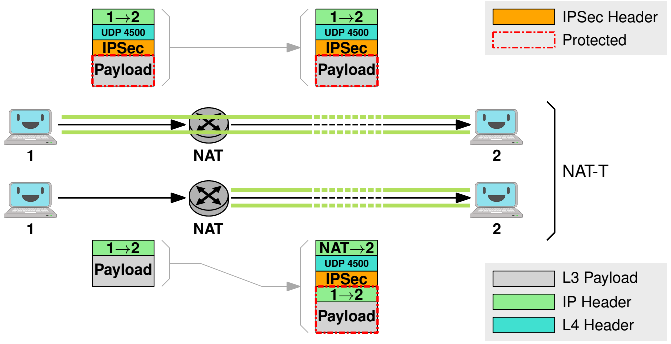


Secure Channels: IPsec

IPsec and NAT

As usual, introducing security introduces further problems . . .

- IPsec hides (encrypts) the L4 protocol
  - ⇒ The L4 header is encrypted
  - ⇒ We cannot see the port numbers
  - ⇒ NAT stops working as it relies on port numbers
- Solution: NAT-T (NAT traversal)
  - ⇒ add a dummy UDP header (typically port: 4500)
  - The header has no other purpose than enabling NAT
- The other side just discards the UDP header
- With this dummy header, NAT can work as usual



Secure Channels: IPsec

Wrapping Up

- IPsec enables policy based processing of traffic
  - It enables the enforcement of complex security policies
- It enables confidentiality, authenticity and integrity on Layer 3

- IPsec integrates seamlessly into the network stack
- Packets passing Layer 3 are matched against the policy installed

- IPsec is mostly found in site-to-site tunnels
- For roadwarrior[2] setups traditional VPNs as OpenVPN or Wireguard are more common

[2] A setup in which the client side of the tunnel is on a mobile device (such as a laptop or phone)

Chapter 6: IT Security

Key-Establishment Protocols

The symmetric key protocols we know so far rely on shared secrets. We need a way to dynamically establish such secrets on the fly. This is the task of key-agreement protocols.

Commonly, cryptographic schemes we find in the real world use:

- Long-term (static) keys for authentication
- Short-term (ephemeral) keys lasting only for a session for confidentiality and integrity
  - → These short-term keys are created through key-agreement

Example: **TLS and Certificates**

- TLS implements exactly this split
- Long-lived certificates for authentication
- Short-term keys per session for encryption and integrity protection

Consequences of Loss

? But what happens if the long-lived key/certificate is lost?
  - ! Since this scenario is not unlikely, we need protocols to be robust against it
  - → This is where key-establishment comes into play

Key-Establishment Protocols

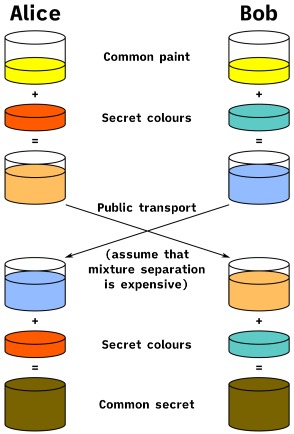
The Relevance of Shared Secrets

Key-Establishment Protocols (aka Key-Agreement Protocols)

- The protocols we have seen so far rely on shared secrets
- We need some way to establish such shared secret
  - ! This has to work over an attacker monitored, insecure channel
  - ! We can't really protect secret establishment as we have no common basis to en-crypt with

- Introducing: key-establishment protocols
- In this lecture we will limit ourselves to the basic Diffie-Hellman key exchange (DH)

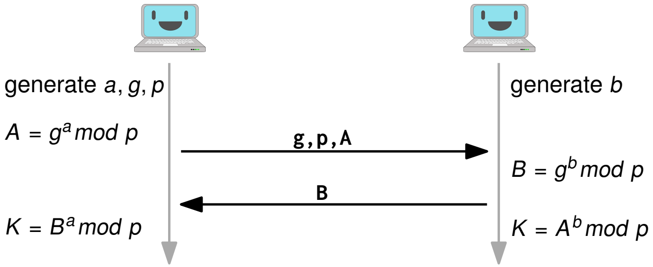
Figure 2: Abstract DH Key-Establishment
Source: Wikipedia, Common Domain



Key-Establishment Protocols

The Diffie-Hellman Key Exchange

- We want to establish a common secret
  - This secret should still be confidential if an attacker eavesdrops on the establishment
- Since paint is messy, we implement the scheme from the previous slide using maths



$$K = A^b \mod p = (g^a \mod p)^b \mod p = g^{ab} \mod p = (g^b \mod p)^a \mod p = B^a \mod p$$

- Knowing g,p,A,B does not help an attacker as they are missing a and b

Key-Establishment Protocols

DH using asymmetric key pairs

The DH key-exchange can also be done on the basis of asymmetric key pairs. This is what you will primarily encounter in the real world.

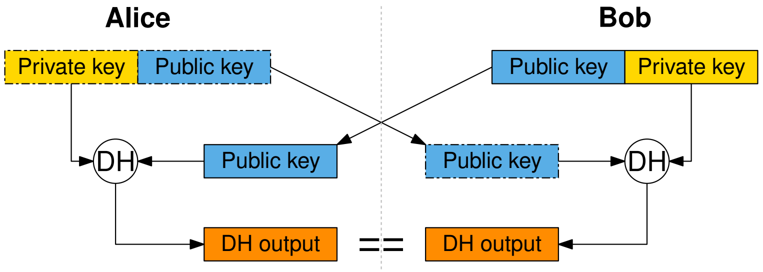


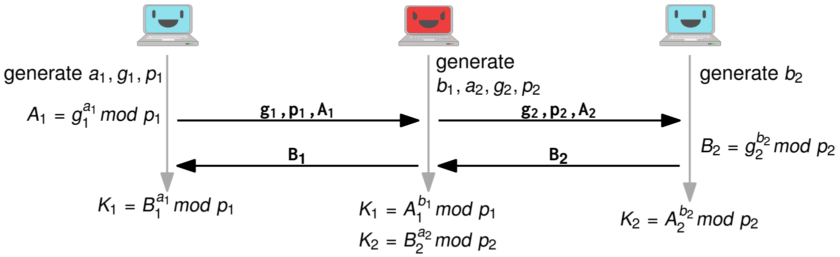
Figure 3: Public Key DH, Source: https://signal.org/docs/specifications/doubleratchet/

- This is quite handy, since we have key pairs anyways
- ! You should use ephemeral key pairs for the DH, otherwise you end up with the same secret every time
- E. g. the Signal Protocol generates ephemeral key pairs frequently and establishes message keys using the former

Key-Establishment Protocols

When Reality sets in

- As usual, reality is a bit more complex
- The standard DH we just presented is easily Man-in-the-Middle'able (the same way the asymmetric encryption scheme was)



generate a_1, g_1, p_1
$A_1 = g_1^{a_1} \text{ mod } p_1$
g_1, p_1, A_1
B_1
$K_1 = B_1^{a_1} \text{ mod } p_1$
generate b_1, a_2, g_2, p_2
g_2, p_2, A_2
B_2
$K_1 = A_1^{b_1} \text{ mod } p_1$
$K_2 = B_2^{a_2} \text{ mod } p_2$
generate b_2
$B_2 = g_2^{b_2} \text{ mod } p_2$
$K_2 = A_2^{b_2} \text{ mod } p_2$

⇒ We need sender authenticity in the DH scheme

- We accomplish this by signing the DH parameters exchanged with a trusted long-term key
  - ⇒ This way, we know that the parameters are from the party we believe they come from
  - → This signing is commonly implemented on the basis of certificates
- DH is also possible using elliptic curve cryptography, known as **ECDH**

Key-Establishment Protocols

- Key-agreement protocols allow us to establish a common secret over an insecure channel
  - → Assuming the existence of a long-lived key, we can also ensure authenticity and integrity of the key-establishment
- We can use such protocols to establish a secret on a per-session basis — the short-lived or ephemeral key
- Having a unique secret per session makes the protocol robust against the loss of the ephemeral key

! We continue by exploring why rotating keys frequently is essential for achieving the desired protocol security properties

Chapter 6: IT Security

Forward and Backward Secrecy

So far, we based all cryptographic properties of protocols on the fact that the static secret remains secret.

Secrets get stolen

- At some point, a secret may get known to an attacker
- The attacker may have eavesdropped on previous traffic
- We would assume that they now can decrypt all previous traffic

! We want cryptography schemes that prevent this

Advancing the Scheme

- We search for a cryptography scheme that provides:
  1. Confidentiality, Authenticity and Integrity
  2. Keeps these properties for already sent messages even when the long-term key or a session is lost

$\Rightarrow$ We obviously cannot use the long-term key for encryption directly

Forward and Backward Secrecy

Who Controls the Present Controls the Past

Forward Secrecy (Perfect Forward Secrecy)
A cryptographic scheme provides Perfect Forward Secrecy (PFS) if previously encrypted sessions maintain their confidentiality in the scenario that the long-term secret, the current session keys and all sessions' traffic become known to an attacker.

- Compromise of the long-term key shall not compromise the short-term keys
- For this to hold, we need the short-term keys to be independent of the long-term keys
- This can be reached e.g. by the means of an ephemeral DH
  - → The authenticity provided through usage of the long-term keys
  - → DH is used to establish short-term keys
- These short-term (ephemeral) keys are then used for encryption
- Compromising the long-term key does not compromise the short-term keys
  - ! Compromising the long-term key still allows the adversary to impersonate the service

Forward and Backward Secrecy

Who Controls the Past Controls the Future

Backward Secrecy (Future Secrecy)
Assume, the attacker has gained access to the long-term secret at point in time `n`. They recorded all traffic and key exchanges afterwards. A protocol provides backward secrecy if messages sent at `t>n` are still secure.

Example: Double Ratchet Algorithm (aka Signal Protocol)

- Based on X3DH (triple DH)
  - Generates keys using a double ratchet mechanism
  - There exists a key for sending and a key for receiving messages
  - These keys are renewed frequently, by deriving a new key from the previous timestamp's key using a key derivation function
  - The keys serve as a basis for a DH, which is used to generate the actual message keys
- Combining key derivation with DH allows to implement Confidentiality, Authentication, Destination Validation[3], Forward Secrecy, Backward Secrecy, Message Unlinkability[4], Participation Repudiation[5] — and a few more properties. [3]

[3] A receiver can prove that they are part of the intended receiver list

[4] Me proving that you authored one message does not allow me to prove that you authored any of the other messages

[5] Given the conversation transcript and all key material except for the accused party I cannot prove that the accused party was part of the conversation

Chapter 6: IT Security

Cryptography Algorithms

- With DH we have seen a key-establishment protocol

→ In this chapter we will look into the details of algorithms for encryption and authentication

Selected algorithms we will discuss:

Asymmetric Cryptography	**Symmetric Cryptography**	**Hashing**	**MACs**
RSA, ECC	AES-{ECB, CBC, CTR, GCM}	MD5, SHA-1, SHA-2, SHA-3, bcrypt, argon2	HMAC

- You will also analyze, attack and then fix some simple crypto protocols in the exercises
- Sadly, reality is not as straightforward
  - ! Most algorithms/ciphers have known weaknesses
  - → Therefore, using them naively will compromise the security of your implementation
  - ⇒ Knowing the details and weaknesses therefore is necessary

Cryptography Algorithms

Asymmetric Cryptography > RSA

Key Generation

```
choose sufficiently large primes p and q
n = pq
$\phi$(n) = (p-1)(q-1)
choose e such that gcd(e, $\phi$(n)) == 1          // e and $\phi$(n) are coprime
choose d such that ed = 1 mod $\phi$(n)

We result with
(e, n) as the public key
(d, n) as the private key
```

Encryption and Decryption

```
encrypt(e, m) = ($m^e$) mod n
decrypt(d, c) = ($c^d$) mod n
```

- The security of RSA is based on the assumption that it is intractable[6] to factorize `n` into `p` and `q`

! In the real world, you will mostly find `e = 65537`. Is this a problem?

- Typical RSA key lengths are 2048 and 4096 bits
  → RSA are larger than symmetric key lengths with the same strength
- RSA's security relies on padding (that is, `m` is sufficiently large)
  - If (m^e) mod n == m^e we have a problem

[6] A problem which can be solved in theory but which in practice takes too many resources to solve.

Cryptography Algorithms

Asymmetric Cryptography > Elliptic Curve Cryptography (ECC)

- Asymmetric Key Cryptography can also be done on elliptic curves
- ECC is based on the discrete logarithm problem
- In contrast to `n` in RSA, clients agree on curve parameters for ECC
  - Different parameters offer different security
  - Therefore there are certified curves, which provide a minimum level of security

- Key sizes for ECC are smaller than for RSA (224 bit offer roughly the same security as 2048 bit RSA keys)
  - Due to the smaller key size the encryption/decryption operations can be done more efficiently

- Quantum computers can factorize large integers with runtime $\in \log(n)$ (Shor's Algorithm)

  $\Rightarrow$ This breaks the basic security assumption of RSA

  :(Sadly, the same holds for ECC

Cryptography Algorithms

Symmetric Cryptography > AES

- AES is just a basic block used to construct ciphers
  - It takes an input block of fixed length and a key and outputs a block of fixed length
  - Without the key the input block cannot be retrieved from the output block
- It can be used to construct block ciphers or stream ciphers

Block Ciphers

- Input is processed in blocks of fixed size
- A block cipher is used to convert a plain text block into a cipher text block
  - The plain text passed through the block cipher
- This means that padding is necessary

Stream Ciphers

- Input is processed byte-wise
- A keystream is generated using the block cipher
  - The keystream can be generated independent of the plain text
  - It is (usually) xor'ed with the plain text to retrieve the cipher text
  - Decryption thereby works in the same way
- The same keystream may **never be used twice**

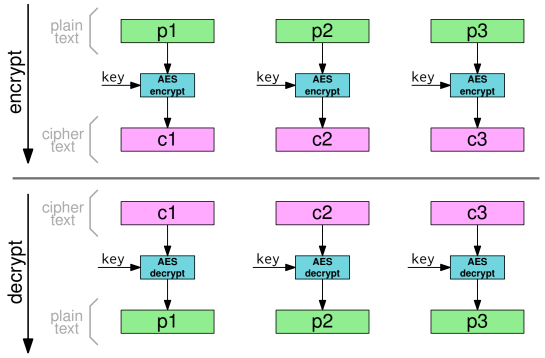
AES Properties

- Key lengths for AES are typically 128, 192 and 256 bits
- The block size is 128 bits, independent of the key length

Cryptography Algorithms

Symmetric Cryptography > AES-ECB (Electronic Codebook)

- A cipher text block depends only on the corresponding plain text block
- This leads to a few unfortunate properties:
    - Same plain text results in same cipher text
    - Patters in the plain text propagate into the cipher text
    - Attackers can reorder, repeat or delete cipher text blocks since the blocks have no cohesion/connection
- Therefore, we only rarely see ECB in real-world use if the plain text is multiple blocks long

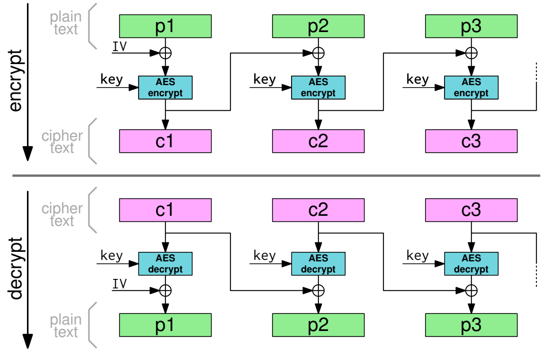


Cryptography Algorithms

Symmetric Cryptography > AES-CBC (Cipher Block Chaining)

- Previous blocks influence later blocks
  - The chaining hides patterns in the plain text

  ⇒ The same plain text does not (necessarily) result in the same cipher text
- Since the first block has no preceding block we use an initialization vector (IV) instead
  - The IV may be public, but it must be unique
- Changing a cipher text block thereby also influences the following plain text block during decryption
- Encryption/Decryption is sequential, therefore it can't be parallelized

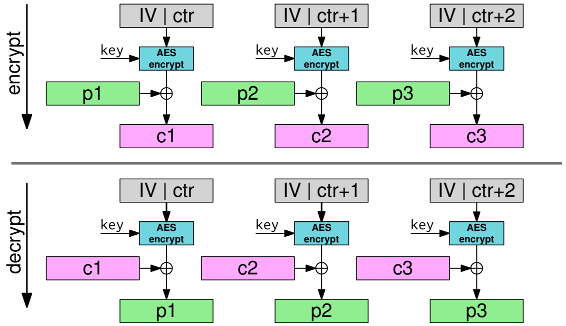


Cryptography Algorithms

Symmetric Cryptography > AES-CTR (Counter Mode)

- Employs AES to construct a stream cipher
- The keystream is generated by encrypting a unique IV concatenated with a counter
  - If the same IV is re-used for the same key, the security is compromised
  - Therefore, if the counter overflows the IV **must** be re-newed or the security of the cipher is compromised
- The keystream can be generated in advance, thereby encryption/decryption can be parallelized
- AES encrypt operations are used for encryption as well as decryption

- AES-CTR allows an attacker to deterministically change the plain text as bits flipped in the cipher text directly propagate
  - If properties of the plain text are known this enables pow-erful attacks
- Therefore, AES-CTR should only be used combined with integrity protection



Cryptography Algorithms

Symmetric Cryptography > AES-GCM (Galois/Counter Mode)

- AES-GCM is an AEAD[1] cipher
- Provides confidentiality as well as authenticity and integrity
- Internally, AES-GCM is an AES-CTR but further processes the input into an authentication tag
  - CTR is combined with the Galois mode of authentication
  - Galois operations can be accelerated by commonly available hardware
- As with CTR, the security depends on the IV being unique (for the same key)
- It has the same parallelization properties as AES-CTR

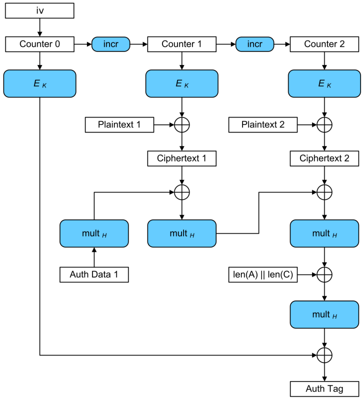


Figure 4: AES-GCM, Source: Wikipedia, CC0

[1] authenticated encryption with associated data

Cryptography Algorithms

Hashing

Internally, Hash functions implement confusion and diffusion properties:

- **Confusion**: Changes in the input shall lead to unforeseeable changes in the output (the relationship is unpredictable)
- **Diffusion**: A bit of the input influences several bits of the output (leading to a complex relationship)

Note: The same properties exist for the algorithm used in a block cipher.

MD5 (1992)(broken)

- Merkle-Damgård scheme
- 128-bit digest from 512-bit input blocks

SHA-1 (1995)(broken)

- Merkle-Damgård scheme
- 160-bit digest from 512-bit blocks

Not so Cryptographic After All

- Both MD5 and SHA-1 are vulnerable to collision attacks [2]
  - Given a input x, it is possible to find a different input y that has the same hash
- Both MD5 and SHA-1 are vulnerable to chosen prefix attacks
  - Given m_1 and m_2, it is possible to find s_1 and s_2 such that hash($m_1 s_1$) == hash($m_2 s_2$)
- Both MD5 and SHA-1 are vulnerable to length extension attacks
- Any of the above three breaks the e. g. signature schemes based on MD5/SHA-1

Cryptography Algorithms

Hashing > SHA-2 (Secure Hash Algorithm 2)(2001)

SHA-2

- The SHA-2 family consists of SHA-224, SHA-256, SHA-384, SHA-512 — all based on the Merkle-Damgård scheme
- They produce a digest of 224, 256, 384 or 512 bits respectively
- The block size is 512-bit for SHA-{224, 256} and 1024-bit for the others

- SHA-{256, 512} are vulnerable to length extension attacks

Merkle-Damgård and Length Extension Attacks

- Merkle-Damgård based hash functions are vulnerable to length extension attacks
  - This is as the internal state length of the hash function is equal to the digest length
- Assume message m and hash $h = \text{hash}(m)$
- h represents (large parts of) the state of the hash function after processing m
- An attacker who knows h thereby can load this state into the hash function and then compute $h' = \text{hash}(m|m')$ without knowing m
- This breaks e. g. the MAC scheme $\text{mac}(\text{key}, x) = \text{hash}(\text{key}|x)$

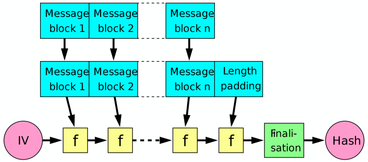


Figure 5: Merkle-Damgård Construction
Source: Wikipedia, Public Domain

Cryptography Algorithms

Hashing > SHA-3 (2015)

SHA-3

- Consists of SHA3-224, SHA3-256, SHA3-384, SHA3-512
  - A subset of the wider Keccak cryptographic primitive family
- Based on a sponge construction
  - The internal state is wider than the digest length
  - This makes it resistant to length extension attacks
- Block sizes vary between 1152 and 576 bits
- As of now, no known attacks that drastically reduce the security

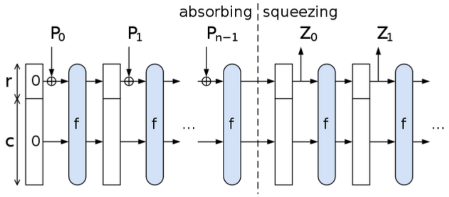


Figure 6: Sponge Construction: Inputs P_i, Outputs Z_i
Source: CC-BY 3.0 `https://keccak.team/sponge_duplex.html`

Cryptography Algorithms

Password Hashing > bcrypt (1999) and argon2 (2015)

Storing Credentials

- When storing passwords, we don't store them in plain text
  - Storing `hash(password)` has a drawback:

  ⇒ Same passwords result in the same hash
- In reality, we therefore use a salt for hashing passwords
  - We rather store `hash(password | salt)`
  - The salt prevents computation of so called rainbow tables[7]
- The `salt` can be public, it remains constant for a single user

  ! It must differ between users

Hashing Passwords should be slow

- Cryptographic password hash functions such as bcrypt or argon2 are configurable
- They have parameters that make them intentionally slow and/or resource intensive
  - This is very much in contrast to the fast and optimized hash functions we have seen so far
  - A slower hashing is acceptable for computing a single hash, but it hurts when you compute a large number of hashes
- Making the computation intentionally slow hinders attackers in brute-forcing passwords and creating rainbow tables

[7] A rainbow table allows looking up an input for a given hash value. Rainbow tables are computed by hashing unique values and storing the relation.

Cryptography Algorithms

MACs, Authentication and Integrity > HMAC

- A MAC (Message Authentication Code) ties the hash of an input to a secret
- Knowledge of the secret and input allows to verify the MAC
- HMAC is a scheme to construct a MAC using a cryptographic hash function

```
HMAC(K,m) = H((K’ ⊕ opad) | H((K’ ⊕ ipad) | msg))

K’ = key if length(key) > block_size else H(key)
ipad = 0x36
opad = 0x5c
                                   where | denotes concatenation
```

Listing 1: HMAC computation

- The transmitted message is `msg | HMAC(key, msg)`
- The receiver verifies that `h == HMAC(key, msg)`

Chapter 6: IT Security

A few final Words

We have gained an insight in basic aspects of IT-Security. We have...

- defined a common language as well as security goals
- mentioned different attack vectors and how they are associated with above goals
- understood the basics of different modes of encryption and the concept of (cryptographic) hash functions
- captured the essential problem of the necessity of a shared secret
- solved the trust problem using trusted third parties and authentication
- learned how digital signatures work in their basic concept
- attacked and fixed simple protocols (see tutorial)
- established how policies and secure channels are implemented by IPsec
- realized the importance of frequent key renewal, and how it is implemented using key-agreement protocols
- understood the meaning of forward and backward secrecy
- looked at detailed properties of different ciphers for encryption, hashes and MACs

I tried so hard and got so far. . .

IT-Security is a complex, rapidly evolving field, and is often neglected in education. Extensive knowledge in this field is usually self-taught.

In this lecture, we only scratched the surface — we have not looked at:

- How to secure information on hosts and in organizations
  - $\rightarrow$ ACLs, MAC and security models
- Network security
  - $\rightarrow$ Network segmentation, firewalls and intrusion detection/prevention systems
- How systems are certified
  - $\rightarrow$ Common criteria, certifications
- Threat actors in reality, their capabilities and evolvement
  - $\rightarrow$ Ransomware, APT, state-level actors and supply chain attacks
- Exploit writing and defense
  - $\rightarrow$ Software vulnerabilities, binary exploitation and limiting damage using isolation and sandboxes
- The relevance of side-channels
  - $\rightarrow$ Meltdown, Spectre, . . .
  - $\rightarrow$ Me extracting your private key using your disk activity LED
- The rapidly rising influence of machine learning and the undefined behavior of the underlying models
  - $\rightarrow$ Attaching critical infrastructure to the one side of the black box that such model is, and the internet to the other, which is fine since nothing bad has ever happened in the internet
  - $\rightarrow$ Explainable Machine Learning

I tried so hard and got so far…

IT-Security is a complex, rapidly evolving field, and is often neglected in education. Extensive knowledge in this field is usually self-taught.

In this lecture, we only scratched the surface — we have not looked at:

- How to secure information on hosts and in organizations
  - → ACLs, MAC and security models
- Network security
  - → Network segmentation, firewalls and intrusion detection/prevention systems
- How systems are certified
  - → Common criteria, certifications
- Threat actors in reality, their capabilities and evolvement
  - → Ransomware, APT, state-level actors and supply chain attacks
- Exploit writing and defense
  - → Software vulnerabilities, binary exploitation and limiting damage using isolation and sandboxes
- The relevance of side-channels
  - → Meltdown, Spectre, …
  - → Me extracting your private key using your disk activity LED
- The rapidly rising influence of machine learning and the undefined behavior of the underlying models
  - → Attaching critical infrastructure to the one side of the black box that such model is, and the internet to the other, which is fine since nothing bad has ever happened in the internet
  - → Explainable Machine Learning

…but in the end it doesn't even matter

Every system you design will have vulnerable points. You may try to limit them, but some remain. Some of them are humans. By that assumption, using a gun bypasses all of your security mechanisms. But usually, a word document with macros, sent by mail, is enough. **Always stick to what is well known and tested, unless you know what you are doing — usually you don't.**

Don’t give up. Just try harder.

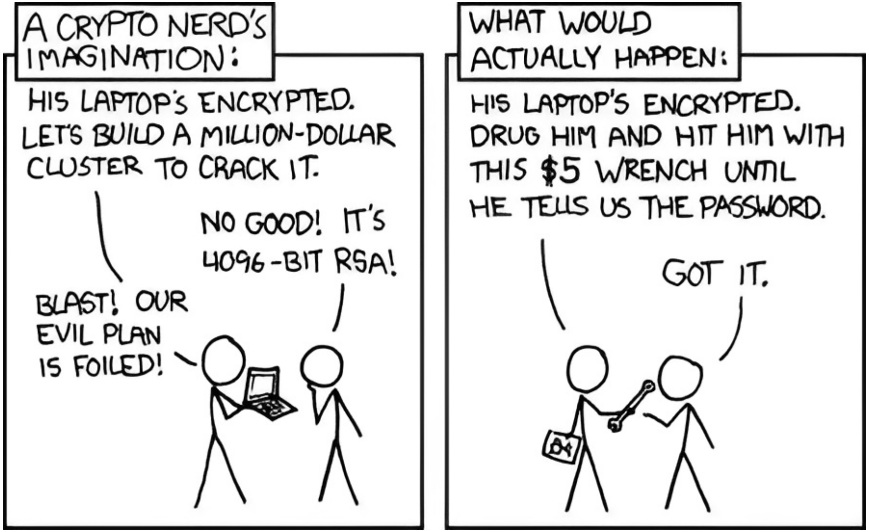

A CRYPTO NERD'S IMAGINATION:
HIS LAPTOP'S ENCRYPTED. LET'S BUILD A MILLION-DOLLAR CLUSTER TO CRACK IT.
NO GOOD! IT'S 4096-BIT RSA!
BLAST! OUR EVIL PLAN IS FOILED!
WHAT WOULD ACTUALLY HAPPEN:
HIS LAPTOP'S ENCRYPTED. DRUG HIM AND HIT HIM WITH THIS $5 WRENCH UNTIL HE TELLS US THE PASSWORD.
GOT IT.

Figure 7: Source: https://xkcd.com/538/

Chapter 6: IT Security

References

[1] *System Administration Guide, Volume 3*, chapter Overview of IPsec, pages 371–373.
Sun Microsystems, Inc., 2000.
https://docs.oracle.com/cd/E19455-01/806-0916/806-0916.pdf.

[2] A. Albertini.
Easy sha-1 colliding pdfs with pdflatex.
In *PoC||GTFO*, page 63, June 2018.

[3] N. Unger, S. Dechand, J. Bonneau, S. Fahl, H. Perl, I. Goldberg, and M. Smith.
SoK: Secure Messaging.
In *2015 IEEE Symposium on Security and Privacy*, pages 232–249, San Jose, CA, USA, May 2015. IEEE.