CRYPTOGRAPHY AND KEY MANAGEMENT

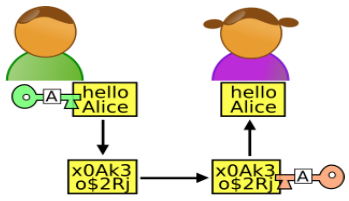
UTM 2024, Marius Rogobete

Agenda

TERMINOLOGY AND BASICS

CRYPTOGRAPHY - DEFINITION

- **Cryptography (AKA “encryption”) is a set of techniques & computations that are applied for securing data and communication against interception and eavesdropping.**
- **It is the most important tool for enhancing Cyber-security and privacy, *<u>when it is applied in a correct way</u>***
- **We use it daily in our phones, our web surfing, when we travel (in epassports), when we use public transportation tickets, when we pay with credit cards etc.**
- **This training is focused on its applications in the embedded context.**

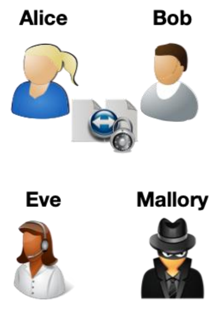


TERMINOLOGY & BASICS

THE ECOSYSTEM

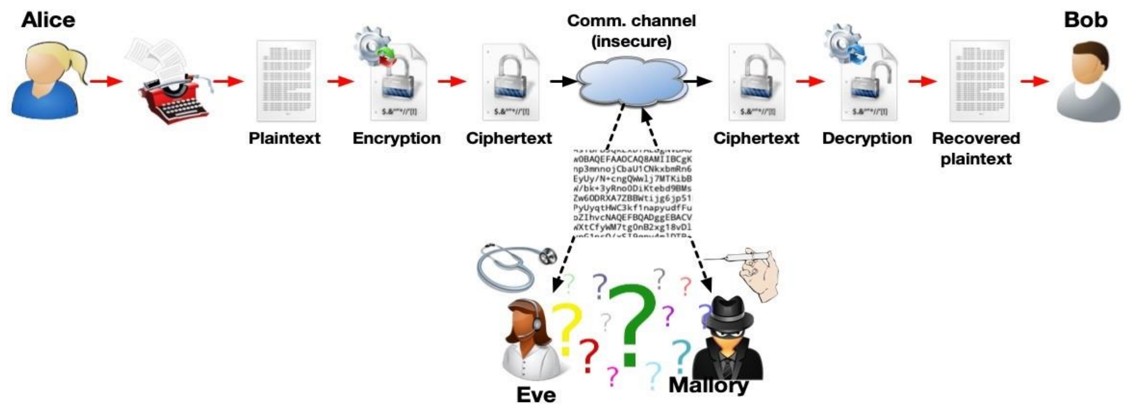
The cryptographic ecosystem:

- **2 parties (A & B) that want to communicate with confidentiality, often called “*Alice*” & “*Bob*”**
- Sometimes we may add “*Charlie*”, “*Dave*” & others
- **A passive eavesdropper, often called “*Eve*”**
- **A more active/malicious adversary, often called “*Mallory*”**
- **The message that needs confidentiality – the “*plaintext*”**
- **The encrypted message – the “*ciphertext*”**
- **The encryption/decryption process (“*algorithm*” or “*protocol*”)**



TERMINOLOGY & BASICS

THE ECOSYSTEM – CONT.



TERMINOLOGY & BASICS

USE CASES

Cryptography is applied for other use cases, not only for confidentiality:

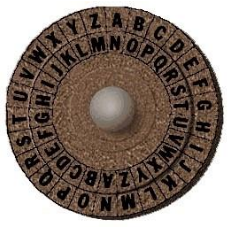
- **Integrity protection** ✓ Examples: data in e-passports, signed software/firmware
- **Identity management & establishing trust** ✓ Examples: TLS, proving website identities to browsers
- **Access control** ✓ Examples: automotive RKE, diagnostics access to ECUs
- **Authenticity** ✓ Example: cryptocurrencies (proving that transactions were committed)

CRYPTOGRAPHY IN HISTORY
(& SOME IMPORTANT LESSONS)

CRYPTOGRAPHY IN HISTORY

HARDWARE-BASED CRYPTOGRAPHY

- The Middle Ages (15th century):
  The Alberti wheel from Italy



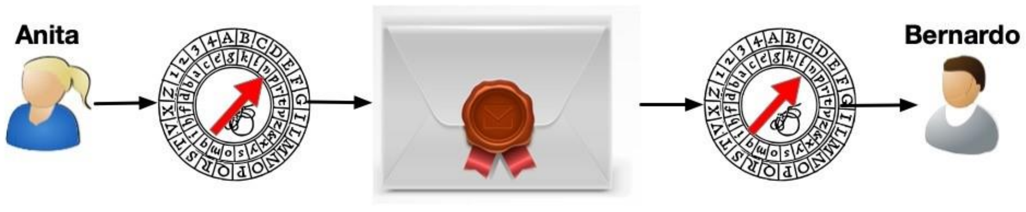
"ALBERTI" → "NYOREGV"

- The 19th century:
  From the USA:

CRYPTOGRAPHY IN HISTORY

AN IMPORTANT ADDITION – THE "KEY"

The Alberti Wheel added something new:



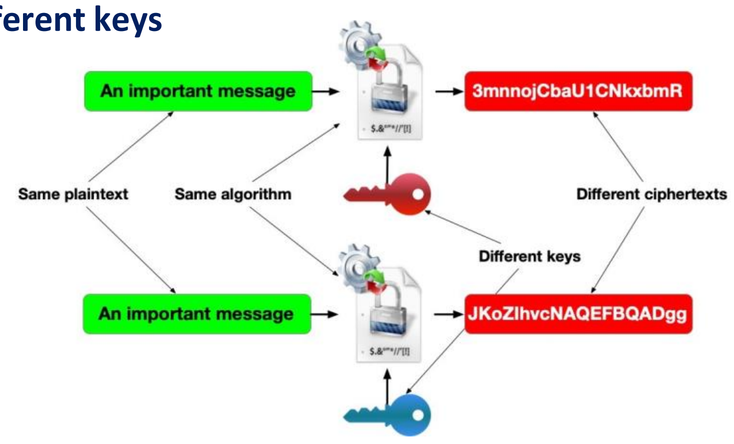
Different relative positions of the inner wheel will produce different ciphertexts

The relative position of the wheels is the "KEY" but the process stays the same:
Substitute each letter in the inner wheel with the adjacent letter in the outer wheel

CRYPTOGRAPHY IN HISTORY

TERMINOLOGY

- We call the cryptographic process"*Algorithm*" or "*Protocol*"
- We differentiate between instances of the same algorithm with different keys



13

13

CRYPTOGRAPHY IN HISTORY

ALSO, IN THE 19TH CENTURY

Kerckhoffs' Law (from 1883):

- The algorithm can be public
- Only the key must be secret
- The key can be changed, transported and stored
- The algorithm may be applied to all relevant data types– Text, maps…
- The process should be easy (complexity ≠ security, fewer logistics = better security)

14

14

CRYPTOGRAPHY IN HISTORY

HOW CAN WE BREAK A CIPHER?

- **There is one attack method that will always succeed:** ***Brute-force attack***

We can try every possible key until we hit something meaningful. Such exhaustive searches over the entire key space are called "*brute force attacks*"

- **Brute force attacks will always succeed, unless we make them impractical**

- Impractical = too long = until the sun runs out of hydrogen
- Impractical = not enough silicon on earth for building HW that will do it
- Impractical = not enough energy on planet earth for this task
- Impractical = too long = for the lifespan of a device
- **We want brute force attacks to be expensive enough for our adversaries**

15

15

CRYPTOGRAPHY IN HISTORY

HISTORY CAN TEACH US MANY LESSONS

- **We always want robust ciphers, that cannot be cryptanalyzed**
- **How do we know what can be trusted?**
- **This is not easy, better left to professional specialist mathematicians**
- **It is far easier to know what CANNOT be trusted, especially when the author claims that the proposed scheme is unbreakable…**
- **We call such bogus products "*Snake Oil*" and there are some telltale signs:**
- **1st and foremost → not in the common standards**
- **Unreasonable key sizes ("128 bits is weak, we use a key with ten thousand bits")**
- **Claiming "military-grade" encryption**
- **No peer review, "trust us, we know what we do"**
- **Algorithm not disclosed (remember Kerckhoff?)**

16

16

THE ZIMMERMAN TELEGRAM –WW1, 1917

WESTERN UNION TELEGRAM

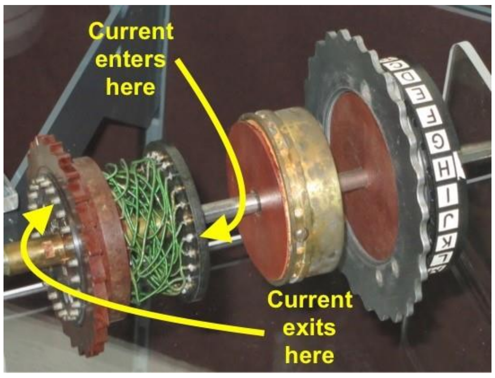
- **In the 19th Century the USA conquered from Mexico large territories in Texas, Arizona & New Mexico**
- **USA was neutral when WW1 started, but still sold arms to the Allies**
- **Germany wanted to keep USA neutral but stop delivery of arms to the Allies**
- **German Minister of Foreign Affairs, Arthur Zimmerman, sent a telegram to the German ambassador in Mexico, promising to let Mexico reclaim those territories if they start a war with USA**
- **The telegram was intercepted and deciphered by the UK**
- Breaking a cipher is called **"*cryptanalysis*"**
- **The decrypted telegram was disclosed to the USA and convinced the USA to join the Allies!**
- **A single event of cryptanalysis that changed the course of history**

17

17

OPERATION "ULTRA"

- **WW2 – the German army used a sophisticated cipher machine called "*Enigma*"**
- **It was based on mechanical/electrical rotors with crisscross wiring:**



18

18

CRYPTOGRAPHY IN HISTORY

THE ENIGMA MACHINE

External view

Under the hood

The internal wiring of the wheels and their order was changed regularly (i.e., different keys)

19

19

CRYPTOGRAPHY IN HISTORY

THE ENIGMA MACHINE

- Originally built for banks
- 3 rotors with ~158,000,000,000,000,000,000 combinations, a 4th rotor was added later
- Initial cryptanalysis started in Poland and results were shared with the UK, including a crude design of a cryptanalysis machine
- The German generals used a different machine with twelve rotors - the *Lorenz* machine

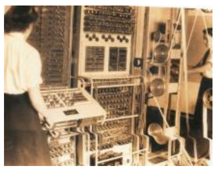
A single case of operator's mistake enabled cryptanalysis of the Lorenz machine

20

20

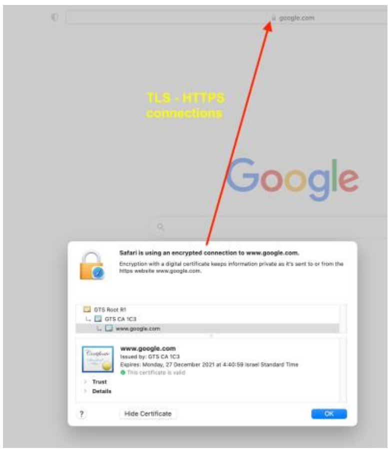
OPERATION ULTRA

- The UK perfected the Polish cryptanalysis machine and were able to decrypt the German encrypted traffic regularly
- The British machine (“*COLLOSUS*” and later “*BOMBE*”) was built in Bletchley Park
- There was also a dedicated machine for the Lorenz traffic
- Alan Turing was a key figure in this operation
- The outcome of WW2 would be very different without *ULTRA*



MODERN TIME

- Today cryptography happens behind the scene in many places, such as web browsers:



Safari is using an encrypted connection to www.google.com.
Encryption with a digital certificate keeps information private as it's sent to or from the https website www.google.com.

GTS Root R1
└ GTS CA 1C3
  └ www.google.com

www.google.com
Issued by: GTS CA 1C3
Expires: Monday, 27 December 2021 at 4:40:59 Israel Standard Time
This certificate is valid

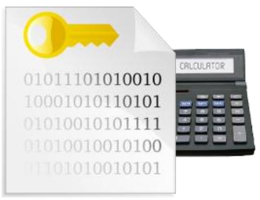
SYMMETRIC CIPHERS

23

SYMMETRIC CIPHERS

A SIMPLE EXAMPLE

- **Simple modulo addition as the algorithm:**
  ○ Encryption algorithm: Ciphertext = Plaintext + Key MOD 10 ○
  Decryption algorithm: Plaintext = Ciphertext + Key MOD 10 ○
  Plaintext = “9” ○ Key = “5”
  ○ Encryption: “9” + “5” MOD 10 = “4” ○ Decryption:
  “4” + “5” MOD 10 = “9”
- **The decryption process is identical to the encryption in this case**
- **The key must be pre-shared or agreed between parties**
- **Because both parties must apply the same key**



24

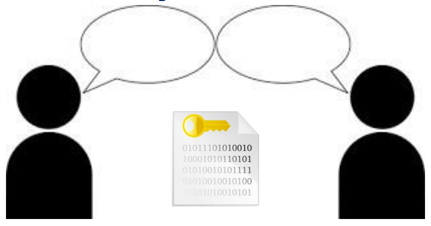
24

SYMMETRIC CIPHERS

A SIMPLE EXAMPLE – CONT.

- In the previous example, both parties applied the same key, with an identical algorithm
- In other cases, the encryption & decryption algorithms can be different but are always related
- Regardless of the algorithms, <u>the same pre-shared key is used by both parties</u>

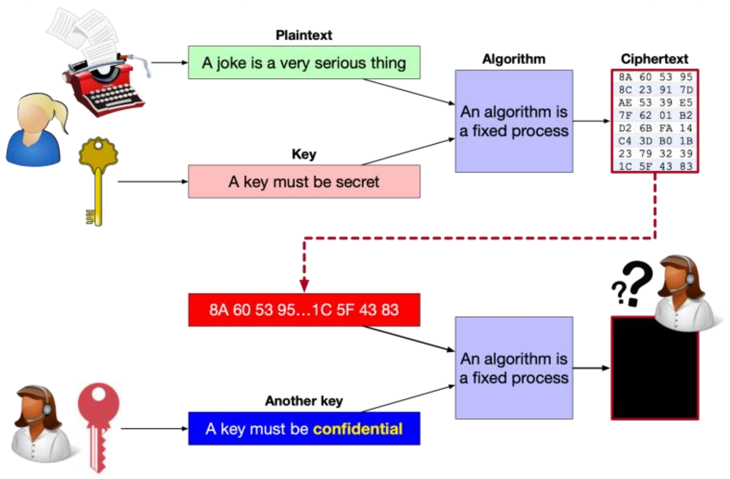
This is called "*symmetric encryption*"



SYMMETRIC CIPHERS

THE GENERIC ALGORITHM

- The algorithm is always a fixed process
- The specific key allows multiple instances, which are opaque to each other



SYMMETRIC CIPHERS

A PRACTICAL ALGORITHM

HARMAN UNIVERSITY

- **Simple addition is obviously not robust enough, we want much better masking of the plaintext and much higher resistance to cryptanalysis**
- **The most common symmetric algorithm today is AES, which is a sequence of very simple operations, that also depend on the key:**
  - XOR with the key
  - Write data in a 4X4 matrix
  - Swap rows & columns in matrix
  - Permutate values (i.e., mix bits in a column from the matrix)
  - Substitute values (i.e., "C" will be mapped to "W")
  - Add/subtract modulo
  - Multiple rounds (10, 12 & 14 for 128. 192 & 256 bits respectively)

D	E	M	O
#	C	I	P
H	E	R	T
E	X	T	#

D	#	H	E
E	C	E	X
M	I	R	T
O	P	T	#

- **Most of the steps use simple lookup tables, so are very fast**
- **Other common algorithms: Triple-DES, Twofish, RC4, XTEA…**

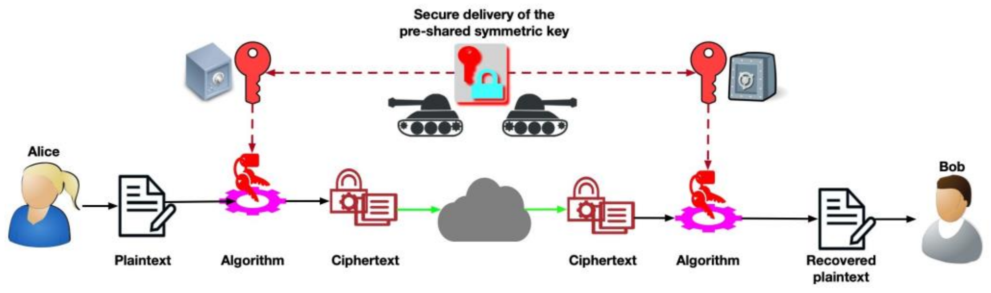
27

27

SYMMETRIC CIPHERS

SYMMETRIC CIPHERS

- **As explained earlier, Alice & Bob use the same pre-shared key for encryption & decryption**
- **"Same pre-shared key" = a <u>secure</u> process to share this key between them before they can communicate with confidentiality**
- **"*Symmetric ciphers*" can be used after <u>both</u> parties hold the same key**



28

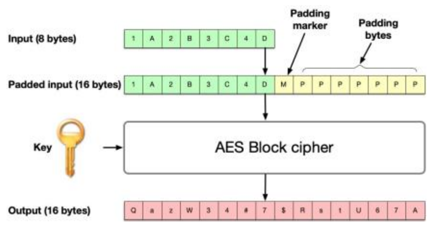
28

BLOCK & STREAM CIPHERS

29

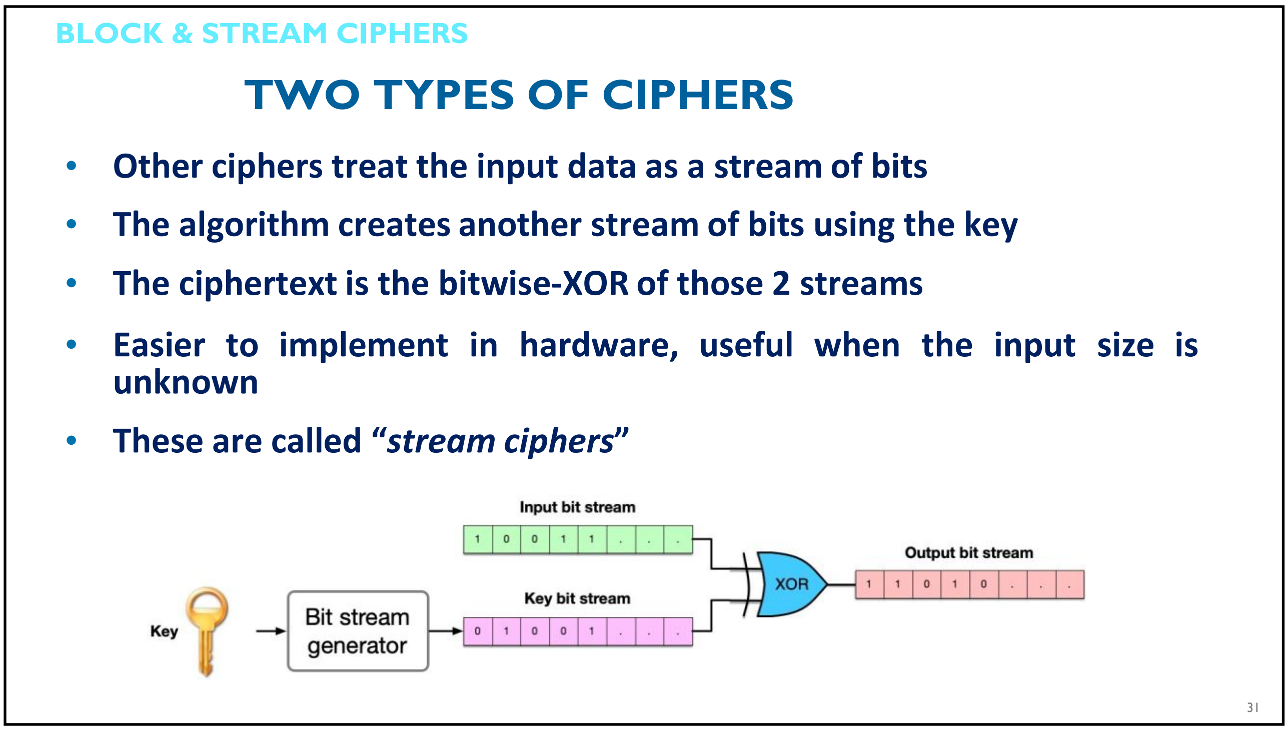
TWO TYPES OF CIPHERS

- Ciphers should be able to handle multiple types of data (remember Kerckhoff?)
- Some types are streams of bits, other types are packets with fixed sizes
- Most ciphers operate on a fixed-size block
- AES uses a block size of 16 bytes, for the input & the output
- If the input is smaller – it will be padded to bring its size to 16
- If the input is larger – it will be split into 16-bytes blocks and each block will be processed on its own (disclaimer: this is only partially true in practice, explained in the following section)
- These are called “block ciphers”



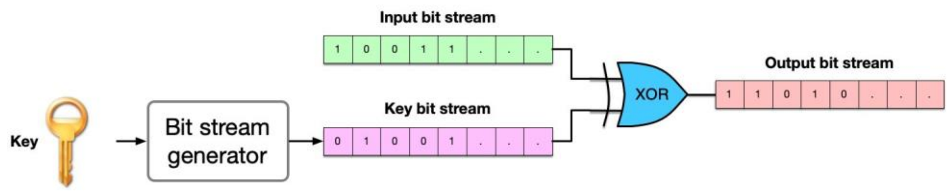
30

30



TWO TYPES OF CIPHERS

- Other ciphers treat the input data as a stream of bits
- The algorithm creates another stream of bits using the key
- The ciphertext is the bitwise-XOR of those 2 streams
- Easier to implement in hardware, useful when the input size is unknown
- These are called "*stream ciphers*"



31

BLOCK CHAINING & MESSAGE AUTHENTICATION

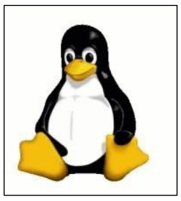
32

BLOCK CHAINING

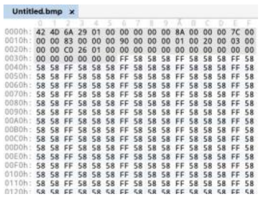
THE “NAÏVE” WAY TO ENCRYPT

- Splitting the input into fixed-size block and processing each block on its own will still leak some information on the input!
- This happens because the same block of plaintext will always produce the same block of ciphertext
- This process is called “ECB” = <u>E</u>lectronic <u>C</u>ode <u>B</u>ook
- Example: encrypting a bitmap:

Original image

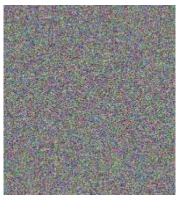


BMP file

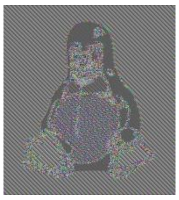


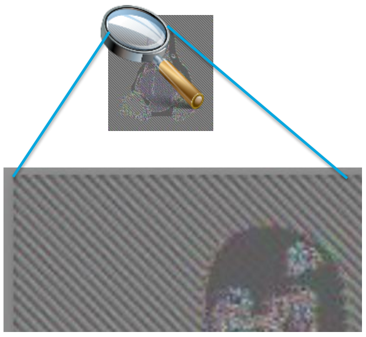
```
Untitled.bmp ×
       0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
0000h: 42 4D 6A 29 01 00 00 00 00 00 8A 00 00 00 7C 00
0010h: 00 00 83 00 00 00 90 00 00 00 01 00 20 00 03 00
0020h: 00 00 C0 26 01 00 00 00 00 00 00 00 00 00 00 00
0030h: 00 00 00 00 00 00 FF 58 58 58 FF 58 58 58 FF 58
0040h: 58 58 FF 58 58 58 FF 58 58 58 FF 58 58 58 FF 58
0050h: 58 58 FF 58 58 58 FF 58 58 58 FF 58 58 58 FF 58
0060h: 58 58 FF 58 58 58 FF 58 58 58 FF 58 58 58 FF 58
0070h: 58 58 FF 58 58 58 FF 58 58 58 FF 58 58 58 FF 58
0080h: 58 58 FF 58 58 58 FF 58 58 58 FF 58 58 58 FF 58
0090h: 58 58 FF 58 58 58 FF 58 58 58 FF 58 58 58 FF 58
00A0h: 58 58 FF 58 58 58 FF 58 58 58 FF 58 58 58 FF 58
00B0h: 58 58 FF 58 58 58 FF 58 58 58 FF 58 58 58 FF 58
00C0h: 58 58 FF 58 58 58 FF 58 58 58 FF 58 58 58 FF 58
00D0h: 58 58 FF 58 58 58 FF 58 58 58 FF 58 58 58 FF 58
00E0h: 58 58 FF 58 58 58 FF 58 58 58 FF 58 58 58 FF 58
00F0h: 58 58 FF 58 58 58 FF 58 58 58 FF 58 58 58 FF 58
0100h: 58 58 FF 58 58 58 FF 58 58 58 FF 58 58 58 FF 58
0110h: 58 58 FF 58 58 58 FF 58 58 58 FF 58 58 58 FF 58
0120h: 58 58 FF 58 58 58 FF 58 58 58 FF 58 58 58 FF 58
```

Properly encrypted



With ECB





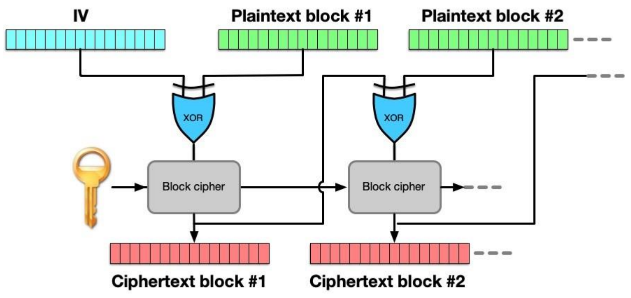
33

33

BLOCK CHAINING

ECB IS BAD, WE MUST DO SOMETHING

- The most common solution is mixing each block with the output of the previous block
- This is called “*CBC*” = <u>C</u>ipher <u>B</u>lock <u>C</u>haining
- The optimal way to mix bits is the XOR function
- The 1st block has no previous block, so we just add an *IV* (= <u>I</u>nitialization <u>V</u>alue) that will be XOR’ed with the 1st block
- The IV must be known to both parties and can be shared in plaintext (i.e., unencrypted)



34

34

BLOCK CHAINING

OTHER CHAINING MODES

- There are other chaining modes, some are focused on specific use cases such as disk encryption, others provide optimized security compared to CBC
- Some sophisticated modes are patented
- **Bottom line: ECB is naïve & insecure, always pick better modes unless you need to encrypt a piece of data that is equal to or smaller than the cipher's block size**

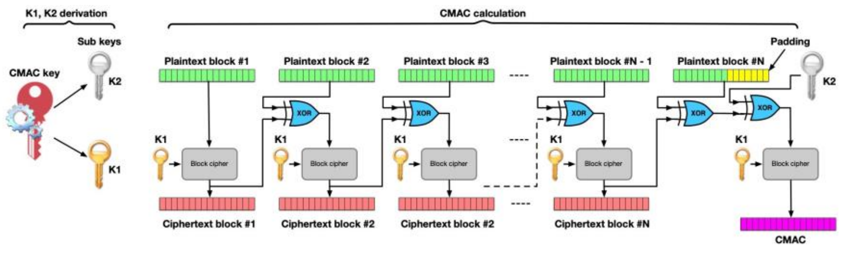
ECB ⛔

CBC, OFB, CTR, GCM, XTS, CFB… ✓

BLOCK CHAINING

MESSAGE AUTHENTICATION

- CBC mode may also be applied for calculating MACs (= Message Authentication Codes)
- Each block depends on the previous block, and the last block depends on the entire message
- We can use the last block as a cryptographic checksum of the message → a MAC
- This is called a CMAC (= Cipher-based Message Authentication Code)
- The real-life CMAC algorithm is slightly different, and we calculate a sub-key that will be XOR'ed with the last message block if it requires padding



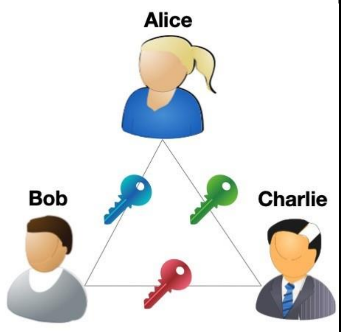
A-SYMMETRIC CIPHERS

37

A-SYMMETRIC CIPHERS

WHAT HAPPENS IF WE HAVE MORE PARTIES?

- **Until now we only had Alice & Bob, with one pre-shared key**
  - **What happens if we add Charlie?**
  - **We need:**
  - one key between Alice & Bob
  - another one between Alice & Charlie
  - yet another one between Bob & Charlie
  - **What happens if we have much more parties?**

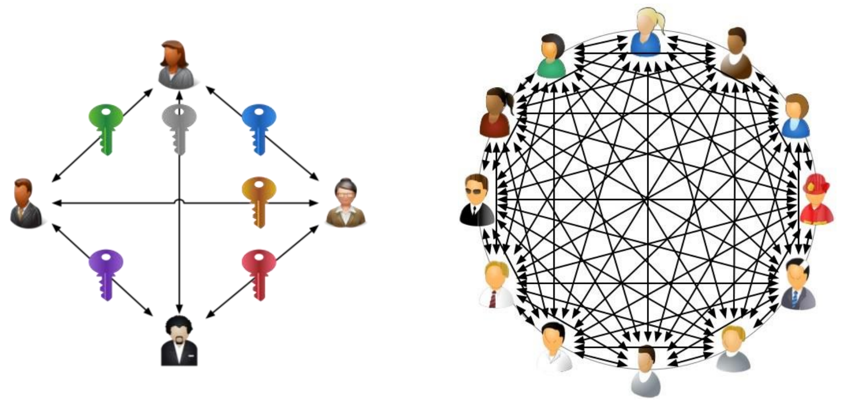


38

38

A SCALABILITY PROBLEM

- We need to pre-share many different keys – a huge logistic hassle

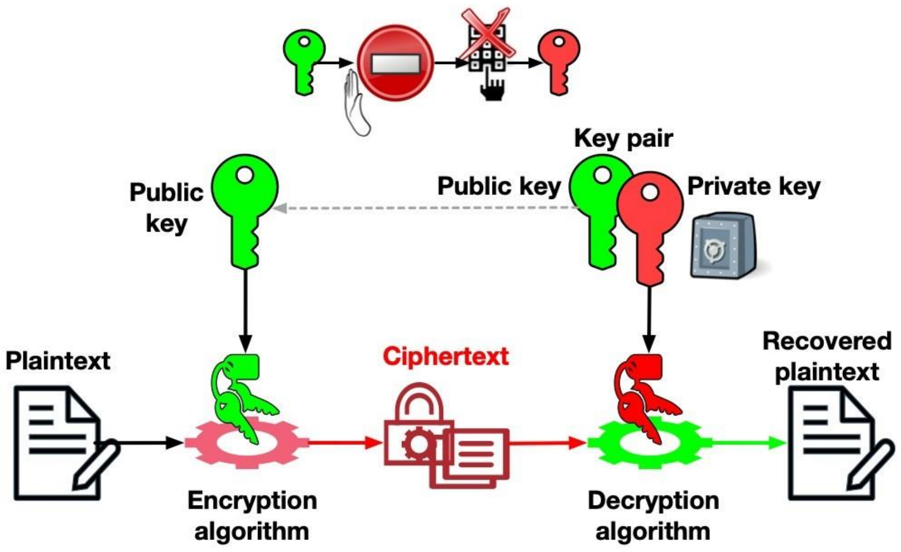


- This is where a-symmetric cryptography enters the stage

A-SYMMETRIC CRYPTOGRAPHY

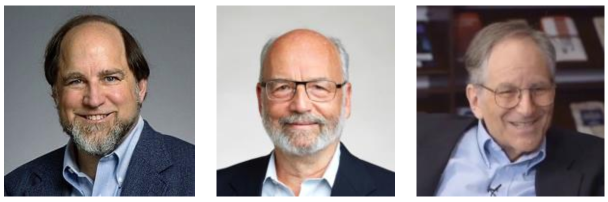
- **We will create an algorithm with 2 separate keys:**
- One (K_E) will be used for encryption
- The other one (K_D) will be used for decryption
- **It should be infeasible to derive $K_D$from K_E**
- May be possible in theory but must be **VERY** computationally-hard
- **Encryption will apply K_E, which can be non-secret (AKA "*public key*")**
- **The encryption should apply a one-way function, so it cannot be reversed by anyone with K_E**
- **Decryption will apply K_D, which should be secret (AKA "*private key*")**

A-SYMMETRIC CRYPTOGRAPHY



A-SYMMETRIC ALGORITHMS

- **This concept was published in the beginning of the 70's**
- **Some cryptographers tried to design such algorithms but failed**
- **A practical algorithm was invented some years later, known as RSA**
- **It is named after the 3 co-inventors:**
- Ron **R**ivest, Adi **S**hamir & Leonard **A**dleman



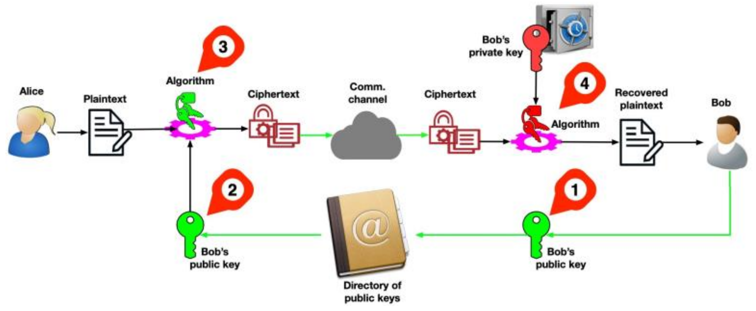
A-SYMMETRIC CIPHERS

A-SYMMETRIC CRYPTOGRAPHY – HOW?

1) **Bob publishes his public key on a public, non-confidential directory**
2) **Alice obtains Bob's public key from this directory**
3) **Alice encrypts her message with Bob's public key**

   Encryption is one-way, so cannot be reversed, **even by Alice**

4) **Bob applies his private key to decrypt the ciphertext and recover the plaintext**



43

43

A-SYMMETRIC CIPHERS

RSA KEY PAIR GENERATION

The a-symmetric RSA key pair is generated using the following method:

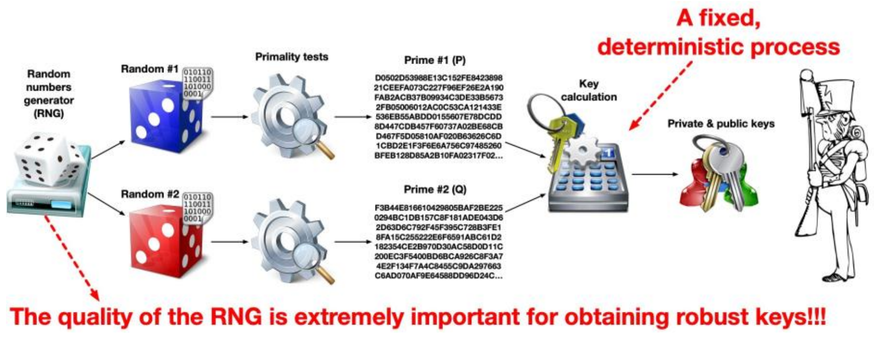
- **2 very big primes, P & Q (hundreds of digits each) are generated**
- **P & Q are then used to derive the RSA key pair (public key & private key) ✓ N = P X Q, the product of the two primes is calculated**
  - ✓ **A prime number E is chosen with no common factor with (P - 1) X (Q - 1) ○ usually, 65537 is used (=2^{16} + 1)**
  - ✓ **The number D is computed with the Extended Euclidean Algorithm (solving an equation for D)**
- **N & E is the public key, N & D is the private key**
- **P & Q are then securely erased (wiped) after key generation, keeping them is risky**

44

44

RSA KEY GENERATION IN PRACTICE

- Practically, we don't really know how to generate very large primes efficiently
- So, we just generate large random numbers and test them for primality
- We do enough tests to detect non-primes with high-enough assurance
- This process is rather slow, even on fast computers


A fixed, deterministic process
Random numbers generator (RNG)
Random #1
Random #2
Primality tests
Prime #1 (P)
D0502D53988E13C152FE8423898
21CEEFA073C227F96EF26E2A190
FAB2ACB37B09934C3DE33B5673
2FB05006012AC0C53CA121433E
536EB55ABDD0155607E78DCDD
8D447CDB457F60737A02BE68CB
D467F5D05810AF020B63626C6D
1CBD2E1F3F6E6A756C97485260
BFEB128D85A2B10FA02317F02...
Prime #2 (Q)
F3B44E816610429805BAF2BE225
0294BC1DB157C8F181ADE043D6
2D63D6C792F45F395C728B3FE1
8FA15C255222E6F6591ABC61D2
182354CE2B970D30AC58D0D11C
200EC3F5400BD6BCA926C8F3A7
4E2F134F7A4C8455C9DA297663
C6AD070AF9E64588DD96D24C...
Key calculation
Private & public keys
The quality of the RNG is extremely important for obtaining robust keys!!!

45

THE RSA PROCESS

- **Encryption: ciphertext = plaintext^E MODULO N**
- **Decryption: plaintext = ciphertext^D MODULO N**
- **Security depends on the difficulty of factoring very large numbers → a very hard task**
- **Brute-force attack: we can try every possible prime in range. More efficient attacks exist, but are still not practical on large enough keys (large enough = 3000 bits or more)**
- Will take many billions of years and will consume too much energy, breaking one 2048-bits key is not practical with current technology
- **Classical computing is not likely to help, quantum computing may become a big risk in the future (future =~ within a decade?)**
- **Break RSA 2048-bits key will require >4000 stable Qubits, state of the art is ~120 Qubits with questionable stability and too short stable lifespan, much less than is required**
- **A breakthrough in quantum computing may happen tomorrow or may not happen at all**

46

A-SYMMETRIC CIPHERS

THE RSA PROCESS - EXAMPLE

- ○ **Primes = 47 & 71**
  - ➢ N = 47 X 71 = 3337
  - ➢ E = 79 (no common factors with 46 X 70 = 3220)
  - ➢ D = 1019 ○ **Plaintext = 688**
  - ➢ Encrypt: PLAINTEXT^{E} MOD N = 688^{79} MOD 3337 =

1570 ○ **Ciphertext: 1570**

- ➢ Decrypt: CIPHERTEXT^{D} MOD N = 1570^{1019} MOD 3337 = 688

47

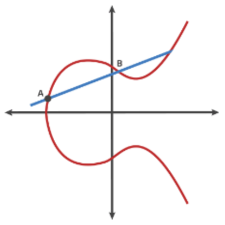
47

A-SYMMETRIC CIPHERS

ALTERNATIVE ALGORITHMS

- **RSA is the most common algorithm, but there are other alternatives**
- **The other alternatives are based on *Elliptic Curves***
- **The elliptic curve is defined by the curve equation**
- Not all elliptic curves are adequate
- The standards define curves that are robust for cryptography
- **The private key is a randomly-chosen point on the curve**
- **Curve equation is used to find another point, which is the public key**
- **Easy to compute when the private point is known**
- **Very hard to compute when the private point is**

unknown • Same security as RSA is achieved with much fewer bits • But very easy to implement incorrectly!



48

48

A-SYMMETRIC SIGNATURES

49

A-SYMMETRIC SIGNATURES

DIGITAL SIGNATURES

- **Until now we had Alice & Bob, looking for <u>confidentiality</u> when they communicate with each other**
- **But when discussing security, we often look for all the CIA properties**
- Not for the famous USA intelligence agency…
- **CIA =** <u>C</u>onfidentiality, <u>I</u>ntegrity, <u>A</u>uthenticity • **What about authenticity and integrity?**
- **Can we achieve them too?**

50

50

A-SYMMETRIC SIGNATURES

DIGITAL SIGNATURES

- A computation that only one specific person can perform is actually a “signature”
- A signature provides authenticity – we know who is the source of a signed again message
- Others should be able to verify it - but not to sign
- **This is where a-symmetric cryptography enters the stage again**
- Encrypting with a PRIVATE key can be done only by the owner of that key
- Everyone else may decrypt with the PUBLIC key
- Correct decryption (i.e., recovering the original plaintext) is a strong proof that the owner of the PRIVATE key did the encryption, if the private key was not compromised
- This is the digital equivalent to signing

[illegible]

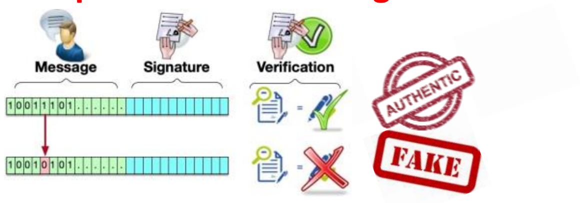
- The plaintext must be known to the verifiers, no confidentiality here

51

A-SYMMETRIC SIGNATURES

DIGITAL SIGNATURES

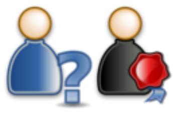
- We already know that a signature provides authenticity
- Remember the CIA paradigm?
- What about integrity?
- This is where a-symmetric cryptography enters the stage again
- Integrity is actually built-in
- Change one bit in the plaintext and the signature verification will fail



52

52

A-SYMMETRIC SIGNATURES



PROPERTIES OF DIGITAL SIGNATURES

- **Digital signatures prove the identity of the signer**
  - ➢ As long as the signer's private key is not compromised
- **Digital signatures are strongly linked to the signed document**
  - ➢ The contents of the signed document cannot be manipulated later
  - ➢ Signature verification will fail if content is modified
- **Digital signatures cannot be transferred to another document**
  - ➢ Unlike graphic signature
- **Digital signatures do NOT protect against cloning**
  - ➢ This requires a dynamic computational process, not a static signature

53

53

A-SYMMETRIC SIGNATURES

DIGITAL SIGNATURES & PKI

- **How do we know that Bob's public key really belongs to Bob?**
- Simple solution: someone we trust will validate it with Bob in person and then sign Bob's public key
- Someone we trust = we know his/her public key with high enough assurance
- **This trusted 3rd party is called a CA (=** Certificate Authority**)**
- **The trustee's signature over Bob's public key is called a "*certificate*"**
- **The collection of technologies we use in this process is called "PKI" (=** Public Key Infrastructure**)**
- **PKI is applied for identity management and for establishing trust**

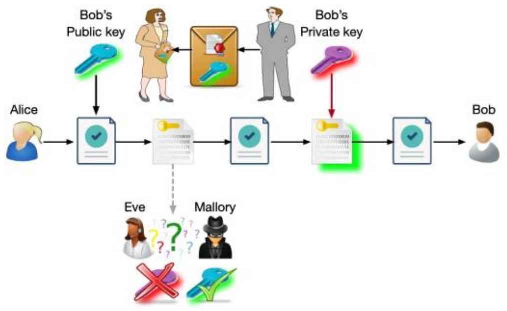
54

54

A-SYMMETRIC SIGNATURES

PKI BASICS

- 1st method: OUT-OF-BAND trust establishment
  - Alice and Bob exchange their public keys in a F2F meeting and then they can communicate securely



  - This works OK but doesn't scale up for multiple parties

55

55

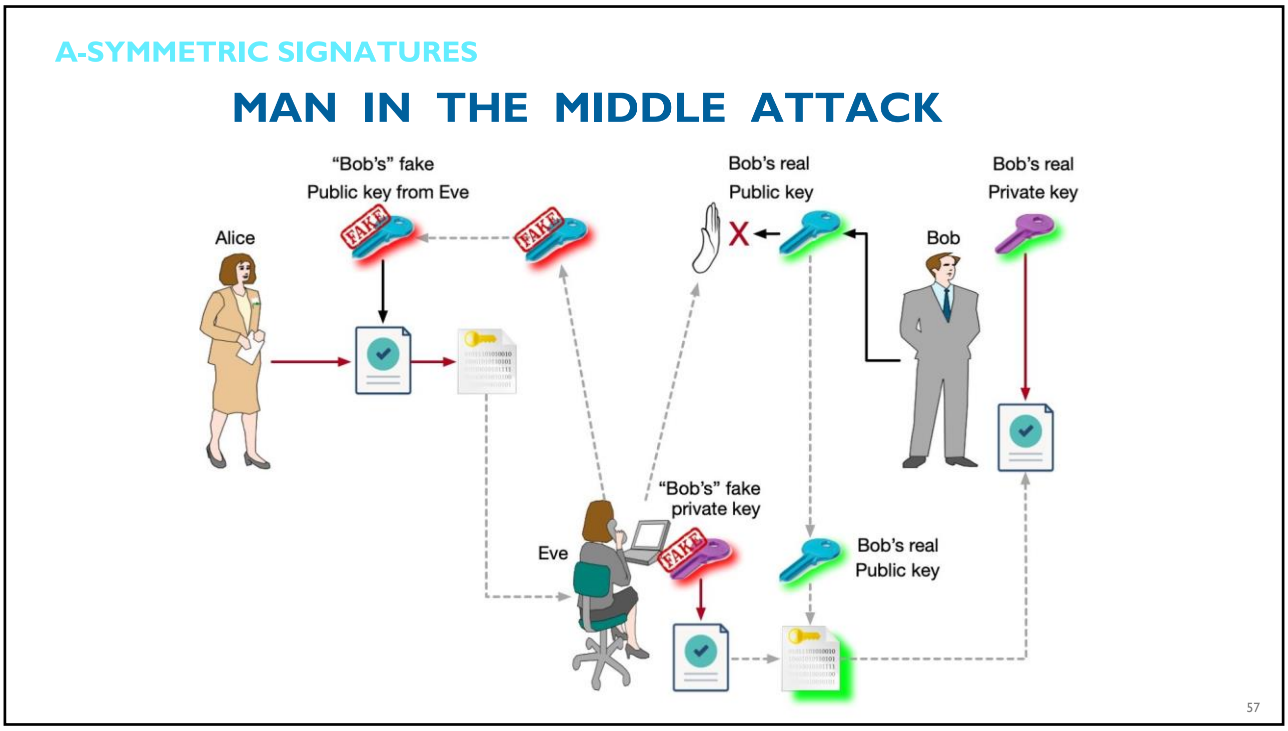
A-SYMMETRIC SIGNATURES

PKI BASICS

- 2nd method: key servers
  - Alice and Bob send their public keys to a server
  - Alice downloads Bob’s public key from the server
  - Alice applies the downloaded public key when she wants to start a secure session with Bob
  - Problem: can we blindly trust what we download from the key server?
  - Can we prevent the “MAN-IN-THE-MIDDLE” attack?

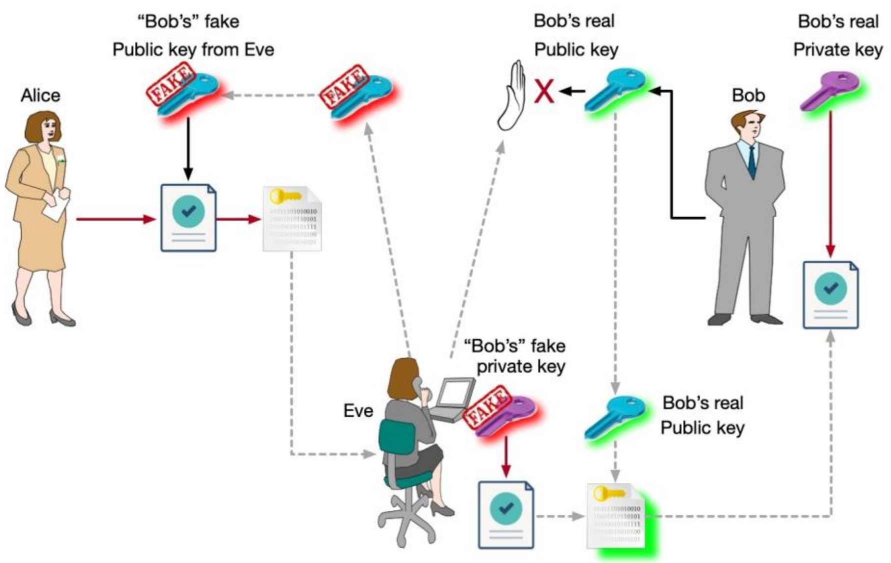
56

56



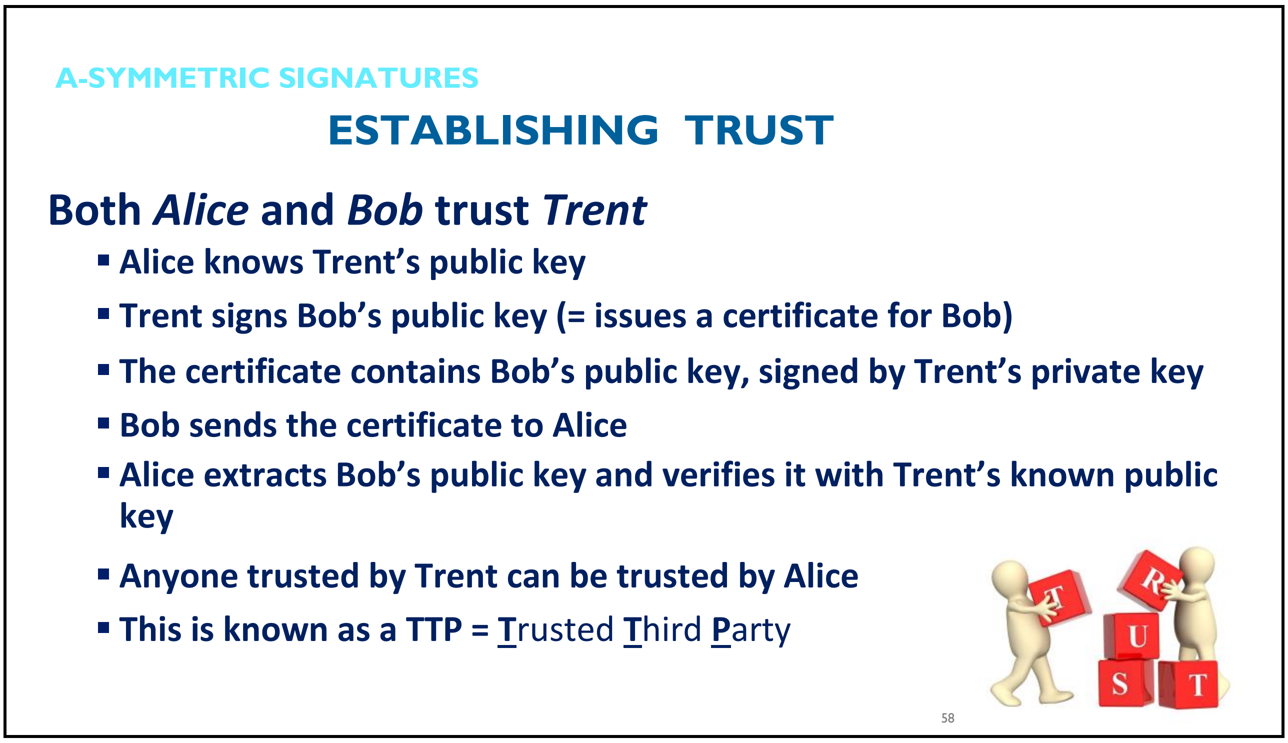
A-SYMMETRIC SIGNATURES

MAN IN THE MIDDLE ATTACK



57

57



A-SYMMETRIC SIGNATURES

ESTABLISHING TRUST

Both *Alice* and *Bob* trust *Trent*

- Alice knows Trent's public key
- Trent signs Bob's public key (= issues a certificate for Bob)
- The certificate contains Bob's public key, signed by Trent's private key
- Bob sends the certificate to Alice
- Alice extracts Bob's public key and verifies it with Trent's known public key
- Anyone trusted by Trent can be trusted by Alice
- This is known as a TTP = Trusted Third Party

58

58

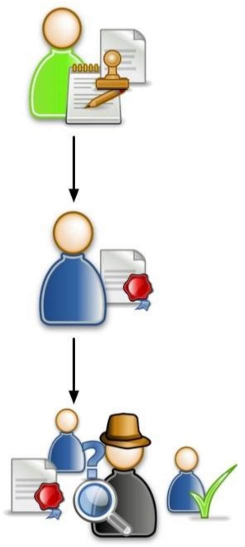
A-SYMMETRIC SIGNATURES

ESTABLISHING TRUST - THE ACTORS

- **Issuer: The entity responsible for issuing the certificate**
  - ➢ **Often called a CA (= Certificate Authority)**

- **Subject: The entity described by the certificate**
  - ➢ **Can be a person, a server or a device**
  - ➢ **Can be a *subordinate CA* if a hierarchy is used** ➢ **Called End Point when it is not a sub-CA**

- **Relying party: An entity that:**
  - ➢ **Wants to verify the subject's certificate/identity** ➢ **trusts the CA**



59

59

A-SYMMETRIC SIGNATURES

CERTIFICATES

- **Certificates are data structures that contain:**
  - ✓ The issuer's public key ✓ The issuer's identity ✓ The subject's public key ✓ The subject's identity ✓ The subject's public key signed by the issuer's private key ✓ Administrative data: issuance date, expiration date etc.
  - ✓ Any data that must be immutable: which actions are allowed with the key?
- **The description of the contents is called a "*certificate profile*"**
- **Certificates contain the PUBLIC key but are not keys!**

60

60

A-SYMMETRIC SIGNATURES

PKI STANDARDS

- **Certificates are data structures, so we need to define the "language" for them**
- **Standards for PKI were defined ~40 years ago**
- **ASN.1 (= Abstract Syntax Notation) was used**
  - ➢ A set of encoding rules defined in the X.690 standard
- **Certificate formats were defined in the X.509 standard, using the X.690 encoding rules**
- **Today probably XML would be used, but it is hard to change well-established standards**

61

61

A-SYMMETRIC SIGNATURES

QUIS CUSTODIET IPSOS CUSTODES?

▪**What about the CA that is the root of trust?** **who signs it?**

➢ The Root CA signs itself

➢ This is called a "***self-signed certificate***"

➢ Result: the Root CA's private key must be well-secured, to prevent a full compromise of the entire trust chain ▪**Root CA security:**

➢ Air-gapped

➢ Physical security

➢ Multi-person control

62

62

A-SYMMETRIC SIGNATURES

REVOCATION

- We know how to establish trust now, but bad things may happen, and keys can be compromised. Can we revoke the trust? How do we do it?
- A method for revoking compromised keys and certificates is definitely needed
- Simple solution:
  - ➢ Periodically publish a list of revoked certificates
  - ➢ This is called a **CRL** (= Certificate Revocation List)
  - ➢ The CRL is signed by the issuer of the certificates
    - ➢ Or by its trusted delegate
  - ➢ We need to revoke certificates not only when a key is compromised
  - ➢ Example: renewing a certificate well ahead of expiration date

63

63

A-SYMMETRIC SIGNATURES

REVOCATION – CONT.

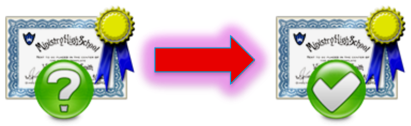
- An alternative method for revoking compromised keys is the *OCSP* method (= Online Certificate Status Protocol)
- An online query is submitted to the OCSP responder • Responder returns the signed status: *revoked* or *valid*
- Much more efficient than CRLs:
  - ➢ Query the status of a single certificate vs. periodically download the entire CRL file, which may become quite large over time
  - ➢ Relevant only when connectivity is available in real time
  - ➢ Response time should be short, but a decent SLA is hard and very expensive when we serve a lot of such queries

64

64

A-SYMMETRIC SIGNATURES

VALIDATING A CERTIFICATE



Correct verification process:

- **Subject** presents its certificate (i.e., claims an identity)
- **Relying party** verifies certificate against trusted root key
- **Relying party** checks that certificate is not revoked (through CRL or OCSP)
- **Relying party** submits a random challenge to **Subject**
- **Subject** signs challenge with its private key as **PoP** (=Proof of Possession)
- Challenge signed correctly → a solid proof that **Subject** really holds the private key → **identity verified with high assurance**

65

65

A-SYMMETRIC SIGNATURES

KEY EXCHANGE

- **2 parties that know each other's public key can exchange encrypted messages**
- **A-symmetric encryption consumes a lot of resources and is not practical when dealing with a lot of data**
- **A more efficient process is required in such cases**
- **A-symmetric encryption can be used to exchange a small symmetric session key, generated by either one of them**
- **The symmetric key can be applied to encrypt large amounts of data with much better performance than a-symmetric encryption**
- **This is called "Hybrid encryption"**

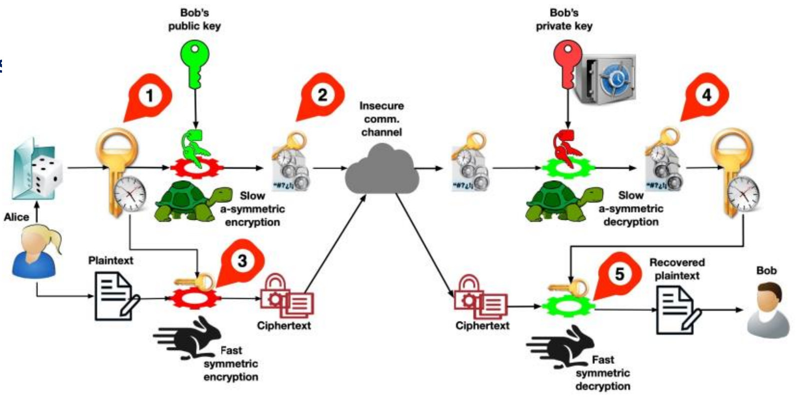
66

66

A-SYMMETRIC CIPHERS

KEY EXCHANGE – HOW?

1) Alice generates a symmetric session key
2) Alice encrypts the session key with Bob’s public key and send it to Bob
3) Alice then encrypts her plaintext message with the symmetric key and sends to Bob
4) Bob applies his private key to decrypt the symmetric session key
5) Bob applies the decrypted symmetric key to recover the plaintext



67

67

A-SYMMETRIC SIGNATURES

KEY AGREEMENT

- A slightly modified process can be used to calculate a shared secret, that will act as the session key, instead of direct delivery of the encrypted session key
- Such a process is called “Key Agreement”
- The most common protocol for key agreement is called “*Diffie-Hellman*” or DH
- DH uses RSA key pairs, ECDH uses elliptic curve key pairs
- A set of parameters are mutually exchanged, including the public keys
- Afterwards each party executes its own calculations with the private key
- Both parties will compute the same shared secret (i.e., the symmetric session key) if they have the correct corresponding private keys
- Being able to communicate with the calculated shared secret is an implicit mutual authentication

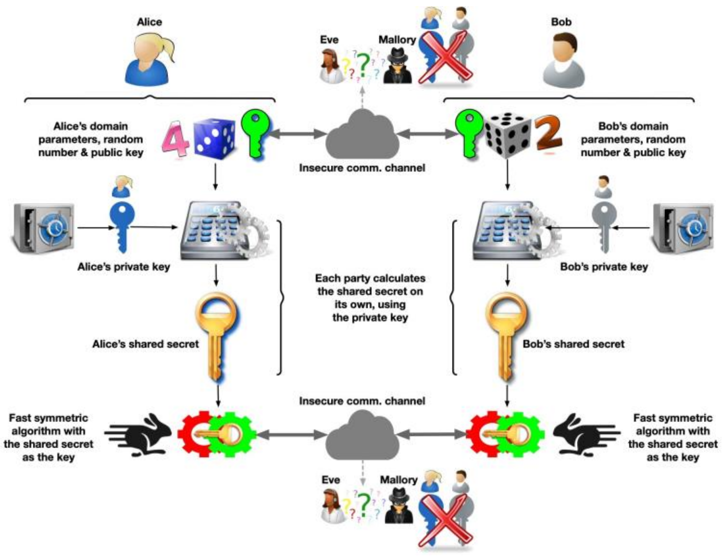
68

68

A-SYMMETRIC CIPHERS

KEY AGREEMENT – HOW?

1) Alice and Bob exchange a set of non-secret parameters
2) Alice and Bob calculate a shared secret using their respective private key
3) The shared secret is applied as a symmetric key between them
4) Eve and Mallory cannot calculate the shared secret because they don't have access to the private keys


Alice
Eve
Mallory
Bob
Alice's domain parameters, random number & public key
Insecure comm. channel
Bob's domain parameters, random number & public key
Alice's private key
Bob's private key
Each party calculates the shared secret on its own, using the private key
Alice's shared secret
Bob's shared secret
Insecure comm. channel
Fast symmetric algorithm with the shared secret as the key
Fast symmetric algorithm with the shared secret as the key
Eve
Mallory

69

69

HASH FUNCTIONS

70

HASH FUNCTIONS

WHAT HAPPENS WHEN WE SIGN BIG FILES?

- **RSA is a block cipher, block size = key size**
- **RSA encryption (signing) is a very computationally-intensive task (raising the message to the power of the key D, which is a VERY big number)**
- **Signing a large file will take ages if we just split it to blocks and process each one of them separately**
- **The solution:**
  - ➢ compress any file to a fixed-size data object and sign only this object
- **This small data object should be very sensitive to its input and even if we change one bit – a lot of bits must change in the output**

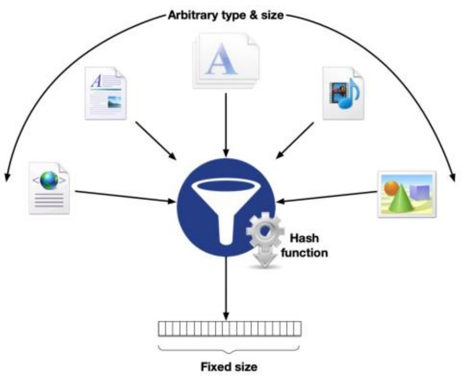
(“pay me 2M$” vs. “pay me 3M$” = one bit difference…)

71

HASH FUNCTIONS

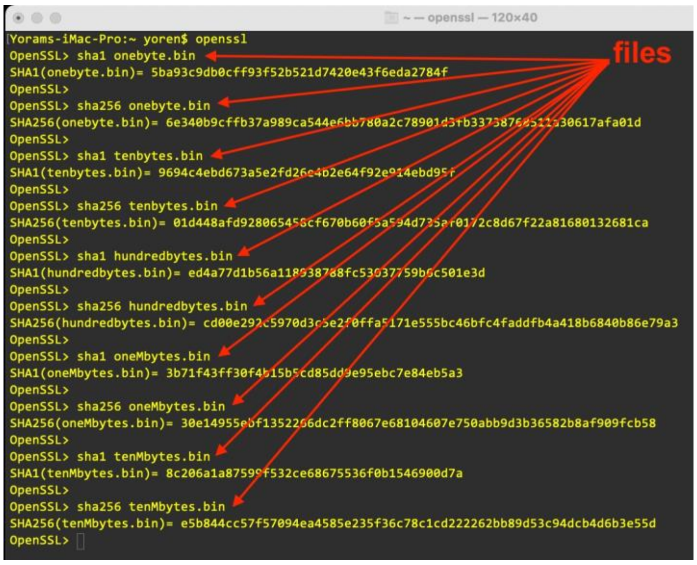
HASH FUNCTIONS

- **This is when hash functions enter the stage**
- **A hash function is a very lossy compression function, also called “Message Digest”**
- **The output is highly dependent on the input and is a very reliable representation of the input**
- **Common hash functions compress data of any size to 128…512 bits**
- **Popular hash functions: MD5, SHA1, SHA256, SHA512, RIPEMD, Keccak (AKA SHA3)**
- **The input may be smaller or larger than the hash size, but the result will always have the same fixed size**
- **A hash can also be combined with a key, to create a secure MAC. This is called HMAC (= Hash-based MAC)**



72

HASH FUNCTIONS - EXAMPLES

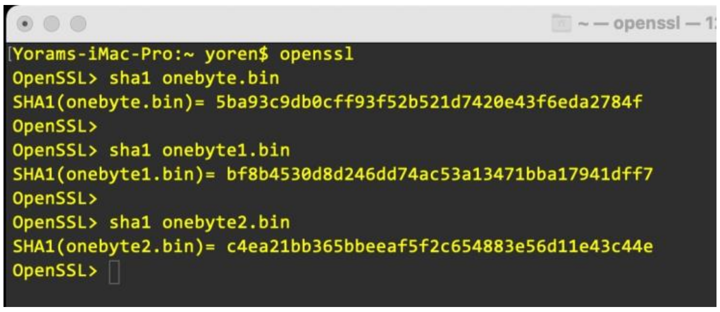


```
Yorams-iMac-Pro:~ yoren$ openssl
OpenSSL> sha1 onebyte.bin
SHA1(onebyte.bin)= 5ba93c9db0cff93f52b521d7420e43f6eda2784f
OpenSSL>
OpenSSL> sha256 onebyte.bin
SHA256(onebyte.bin)= 6e340b9cffb37a989ca544e6bb780a2c78901d3fb33738768511a30617afa01d
OpenSSL>
OpenSSL> sha1 tenbytes.bin
SHA1(tenbytes.bin)= 9694c4ebd673a5e2fd26c4b2e64f92e914ebd95f
OpenSSL>
OpenSSL> sha256 tenbytes.bin
SHA256(tenbytes.bin)= 01d448afd928065458cf670b60f5a594d735af0172c8d67f22a81680132681ca
OpenSSL>
OpenSSL> sha1 hundredbytes.bin
SHA1(hundredbytes.bin)= ed4a77d1b56a118938788fc53037759b6c501e3d
OpenSSL>
OpenSSL> sha256 hundredbytes.bin
SHA256(hundredbytes.bin)= cd00e292c5970d3c5e2f0ffa5171e555bc46bfc4faddfb4a418b6840b86e79a3
OpenSSL>
OpenSSL> sha1 oneMbytes.bin
SHA1(oneMbytes.bin)= 3b71f43ff30f4b15b5cd85dd9e95ebc7e84eb5a3
OpenSSL>
OpenSSL> sha256 oneMbytes.bin
SHA256(oneMbytes.bin)= 30e14955ebf1352266dc2ff8067e68104607e750abb9d3b36582b8af909fcb58
OpenSSL>
OpenSSL> sha1 tenMbytes.bin
SHA1(tenMbytes.bin)= 8c206a1a87599f532ce68675536f0b1546900d7a
OpenSSL>
OpenSSL> sha256 tenMbytes.bin
SHA256(tenMbytes.bin)= e5b844cc57f57094ea4585e235f36c78c1cd222262bb89d53c94dcb4d6b3e55d
OpenSSL>
```

73

73

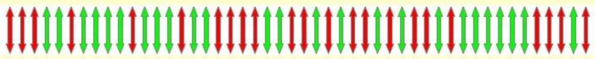
HASH FUNCTIONS - EXAMPLES



```
Yorams-iMac-Pro:~ yoren$ openssl
OpenSSL> sha1 onebyte.bin
SHA1(onebyte.bin)= 5ba93c9db0cff93f52b521d7420e43f6eda2784f
OpenSSL>
OpenSSL> sha1 onebyte1.bin
SHA1(onebyte1.bin)= bf8b4530d8d246dd74ac53a13471bba17941dff7
OpenSSL>
OpenSSL> sha1 onebyte2.bin
SHA1(onebyte2.bin)= c4ea21bb365bbeeaf5f2c654883e56d11e43c44e
OpenSSL>
```

Leading 48 bits of hash result

$$(\texttt{5ba93c9db0cf}\ldots)_{16} = (010110111010100100111100100111011011000011001111\ldots)_2$$



$$(\texttt{bf8b4530d8d2}\ldots)_{16} = (101111111000101101000101001100001101100011010010\ldots)_2$$

23 out of 48 bits changed (~50%), for one bit change in the input!

74

74

HASH FUNCTIONS

- **The compression must be easy to compute**
- **The compression is very lossy, so we DON'T worry about someone being able to reverse it and recover the input**
- **We DO worry about <u>collisions</u>, i.e., finding another input that will produce the same output**
  - A collision allows us to represent something different as the original input–

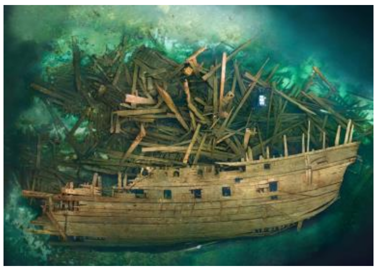
    i.e., counterfeit a document
- **We even worry much more about finding <u>MEANINGFUL</u> collisions**
  - Collisions are useless in real life if they have no meaning
  - However, a meaningless collision still shows us that the hash function is too weak and should be avoided
  - MD5 & SHA1 are considered weak, SHA1 still good for some less-sensitive tasks

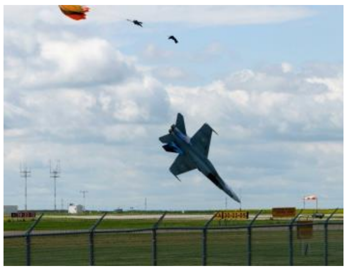
75

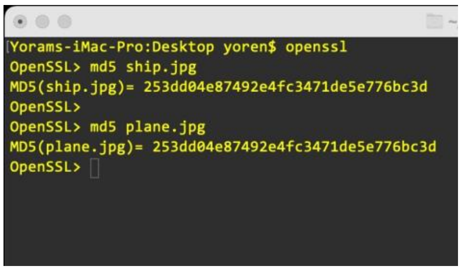
75

HASH COLLISIONS

- **MD5 is an old hash function with 128 bits results**
- **Designed by Ron Rivest in 1991, 1st publicly-known collisions found in 2004**
- **It was found to be <u>very</u> weak and is currently deprecated**
- **There are online databases of known collisions and open-source tools for finding them**
- **Example:**







```
Yorams-iMac-Pro:Desktop yoren$ openssl
OpenSSL> md5 ship.jpg
MD5(ship.jpg)= 253dd04e87492e4fc3471de5e776bc3d
OpenSSL>
OpenSSL> md5 plane.jpg
MD5(plane.jpg)= 253dd04e87492e4fc3471de5e776bc3d
OpenSSL> 
```

from https://natmchugh.blogspot.com

76

76

CRYPTOGRAPHIC KEYS

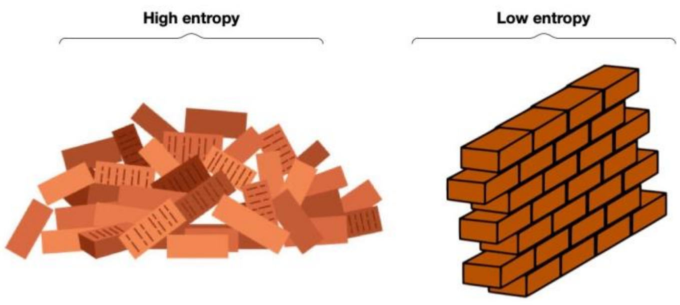
77

CRYPTOGRAPHIC KEYS

- Keys are data objects with some very special qualities, but 1st we must understand one very important term:

Entropy

- Entropy is the amount of disorder, high entropy = a lot of disorder, i.e., more Chaos



78

78

CRYPTOGRAPHIC KEYS

CRYPTOGRAPHIC KEYS

- **If we want our key to be unpredictable or unguessable we want it to be "RANDOM" – or in other words to have high entropy, to be chaotic enough**
- **Obtaining good (= high enough) entropy is a VERY hard engineering challenge**
- A lot of engineers tried and failed
- **A device that provides entropy is called RNG (= R**andom **N**umber **G**enerator**)**
- **Good RNGs are hard to find, we actually settle for "good enough"**

79

CRYPTOGRAPHIC KEYS

GENERATING ENTROPY

- **Good RNGs collect entropy from a source that is:**
  1) Unpredictable: the next bit may be "1" or "0" with a probability of 0.5
  2) Statistically uniform: the probability of "1" = the probability of "0"
  3) Each instance is unique: two instances, placed in the same environment and the same starting conditions, will produce different sequences
  4) High bandwidth: it can produce enough bits per second for the intended use case

- **This is hard to do, mainly because of #4 – good entropy is normally a tradeoff when you consider bandwidth**
- **Good sources of entropy rely on physical phenomena known to be random**
- **Common examples are radioactive decay and Johnson-Nyquist Noise (thermal noise)**

80

80

SOLVING THE BANDWIDTH PROBLEM

- **SW-based RNG is an oxymoron**
- **Good RNGs collect entropy from a hardware-based physical source that provides a few bits of entropy, but bits that are really unpredictable**

  This is called a TRNG (= True **R**andom **N**umber **G**enerator**)**

  ➢ The output of a TRNG may be good but may also be biased, so some post-processing is required to remove this bias and eliminate the risk

- **How do we get higher bandwidth?**

  ➢ We inflate those few unpredictable bits to a larger number of bits

  This is called a PRNG (= Pseudo-**R**andom **N**umber **G**enerator**)**

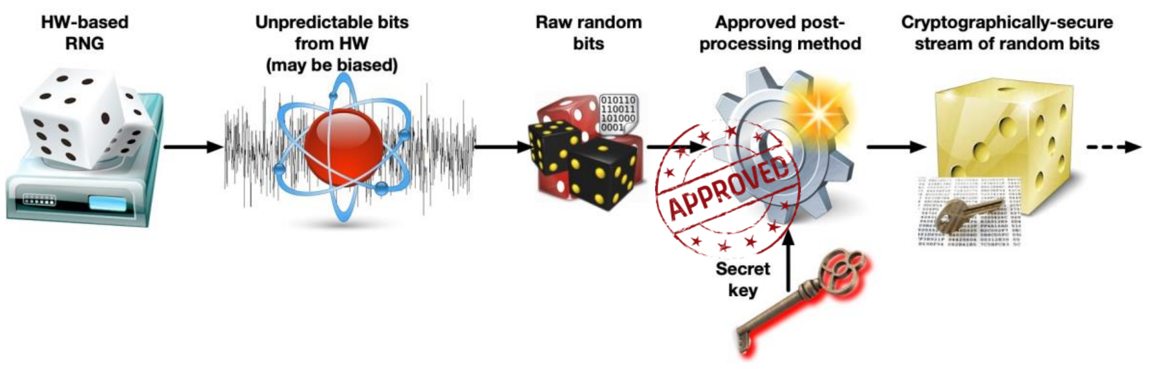
- **Some robust inflation methods allow us to use the PRNG outputs as cryptographic keys**

  This is called a CSPRNG (= Cryptographically-**S**ecure **PRNG)**

81

81

THE GENERIC CSPRNG


HW-based RNG
Unpredictable bits from HW (may be biased)
Raw random bits
010110 110011 101000 0001
Approved post-processing method
APPROVED
Secret key
Cryptographically-secure stream of random bits

82

82

CSPRNG IS ALSO A CHALLENGE

- **3 basic rules for designing a CSPRNG:**
  1) Complexity ≠ Security
  2) There are standards for this, stick to them and never DIY
  3) Don’t break rules 1 & 2
- **Most OSs & programming languages provide a CSPRNG interface**
- **Use the interface provided by a reliable cryptographic library when available**

Linux & ios	/dev/random or /dev/urandom
Windows	BCryptGenRandom from CNG or CryptGenRandom from CAPI
Python	os.urandom & “secrets” module
Java	java.security.SecureRandom

83

83

WHATARETHE QUALITIESWEAREAFTER?

• **We try to build robust systems, well-protected against cyber-attacks** • **Accordingly, our keys must be robust too. They should be:**

✓ **Hard to guess/predict**

–we want keys with the highest entropy possible ✓ **Hard to brute-force**

–Large enough to make brute force too expensive for attackers

✓ **No shortcuts for attackers to exploit**

–Attacks better than brute force should not be possible •

We must generate keys from good entropy sources!

84

84

CRYPTOGRAPHIC KEYS

SO, ALL WE NEED IS A DECENT RNG?

- **A lot of good RNGs exist, few of them are adequate for generating keys!!!!**
- **As we already know, using a standard-based CSPRNG is mandatory**
- **There is no robust process to test random bits, unless we generate a very large block of random bits (at least several tens of megabytes)**
  - ➢ Statistical tests can detect bias but **only if the tested data is large enough**
- **Statistical tests can be used to qualify the RNG but…**
- **We cannot generate a single key and test it for randomness, we must place blind trust on the CSPRNG design and implementation**

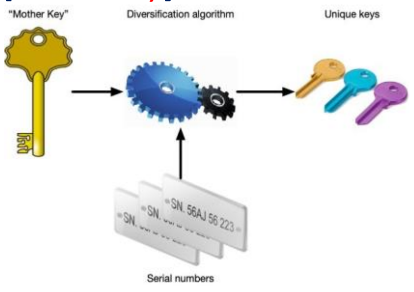
85

85

CRYPTOGRAPHIC KEYS

OTHER KEY GENE ATION METHODS

- **Sometimes we want a key that is que for each instance**
  uni
- **We can generate a lot of random ke ys and keep them in a database but protecting this database is hard to do**
- **Alternatively, we can CALCULATE que keys using a robust protocol**
  uni
- **This is done by mixing a "*Mother* " with a unique identifier of each instance**
  Key
- [*Unique Key*] = [*Unique identifier*] encrypted by [*Mother Key*]
- **This is called "*Key diversification*"**
- **The identifier: ECU barcode, VIN…**



86

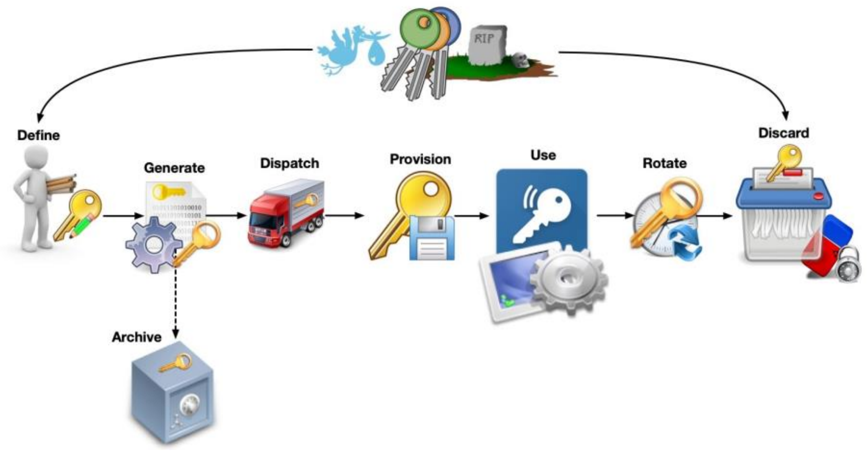
86

KEY MANAGEMENT

87

CRYPTOGRAPHIC KEYS HAVE A LIFECYCLE

Cryptographic keys are objects with a lifecycle:



88

KEY LIFECYCLE - DEFINITION

- **Some key properties must be defined at this phase:**
- **Algorithm – including all applicable data: padding methods, hash functions and chaining mode**
  - ➢ Examples: AES/CBC with PKCS#7 padding, RSA with PKCS#1 V2.1 padding and SHA256
- **Size – expressed in bits**
  - ➢ Examples: 128 bits, 2048 bits, 4096 bits…
- **Formats - how is the key stored or transported**
  - ➢ Examples: DER-encoded, PEM-encoded, PKCS#8, ASCII-HEX…

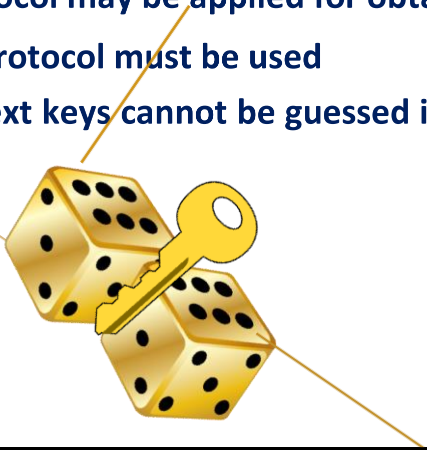
KEY LIFECYCLE - DEFINITION

- **Sensitivity – how attractive is a key to potential adversaries?**
- **Ownership – who owns the key?**
  - – May be managed by X but owned by Y
- **Rollover policy – how often will we update the key?**
- **Generation method**
  - – from a simple RNG if the key is not sensitive?
  - – from a high-grade CSPRNG?
  - – from a key diversification protocol?
- **Validation method – how do we make sure the key is correct?**

CRYPTOGRAPHIC KEYS

KEY LIFECYCLE – GENERATION

- How do we generate a key?
  - Normally a certified CSPRNG is required
  - A key diversification protocol may be applied for obtaining unique keys
  - A robust diversification protocol must be used
  - Robust = not reversible, next keys cannot be guessed if all previous keys are known



91

CRYPTOGRAPHIC KEYS

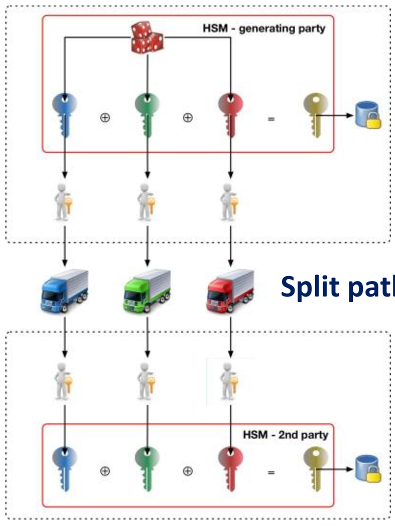
KEY LIFECYCLE –DISPATCH

- How do we deliver the key to its destination?
  - Can we establish a secure channel at all?
  - Do we need face to face delivery?
  - Maybe a secure portal? (try to avoid this!)
  - Maybe use encrypted mail?
  - Or a full-blown Key Ceremony

92

92

KEY CEREMONY

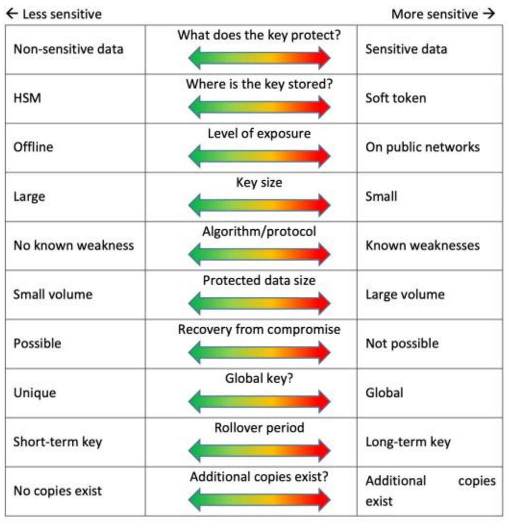


Split path delivery

CRYPTOGRAPHIC KEYS

ASSESSING THE SENSITIVITY

- **How can we know if a specific key is sensitive?**



← Less sensitive		More sensitive →
Non-sensitive data	What does the key protect?	Sensitive data
HSM	Where is the key stored?	Soft token
Offline	Level of exposure	On public networks
Large	Key size	Small
No known weakness	Algorithm/protocol	Known weaknesses
Small volume	Protected data size	Large volume
Possible	Recovery from compromise	Not possible
Unique	Global key?	Global
Short-term key	Rollover period	Long-term key
No copies exist	Additional copies exist?	Additional copies exist

95

95

CRYPTOGRAPHIC KEYS

KEY LIFECYCLE – PROVISION, USE & ROTATE

- **How do we provision a key?**
  - A method for importing a key generated elsewhere is needed
  - On-board key generation may be applied in some cases
  - Attention to residual data in the provisioning tool
- **How do we use the key?**
  - Which protocols apply the key?
- **How often, if ever, will we change the key?**
  - Sensitive keys will be changed more often
  - Not always possible – fused keys are immutable and serve for the entire lifespan

94

94

CRYPTOGRAPHIC KEYS

KEY LIFECYCLE – FROM WOMB TO TOMB

- **The key must be discarded securely when it is not needed anymore**
- **“Not needed” = end of lifespan or just end of session**
- **Simple “delete” is usually not enough**
- **A robust overwrite process must be applied for all copies**
- **Sometimes an expired key may still be a high risk**
  - Especially if adversaries can obtain old encrypted traffic
- **In the embedded context: sensitive key objects must be proactively destroyed after use, good cryptographic libraries do it as a best practice**

96

96

THE EMBEDDED USE CASES

97

EMBEDDED USE CASES

CRYPTOGRAPHY IN EMBEDDED SYSTEMS

There are some common use cases in the embedded context:

- Encrypting personal/private data (example: privacy regulations compliance)
- Protecting intellectual property (example: firmware encryption)
- Content protection (examples: Blu-ray, Playstation games)
- Integrity & Authenticity protection (example: data stored in e-passports)
- Identity management (example: ECU identity in the OEM's cloud)
- Access control (examples: ECU diagnostics, RKE)
- Code signing – signing the firmware against rogue code

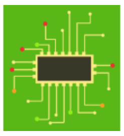
98

98

EMBEDDED USE CASES

FW SIGNING

- FW signing was considered as a best practice, found in high-end platforms • This is becoming mandatory in more & more cases, even in low-end platforms
- We find two different types of FW signatures: ➢ Symmetric FW signing
  - A symmetric algorithm is used to create a checksum over the FW
  - AES-CMAC is very common for this use case
  - The validating key is the same key that is used to sign



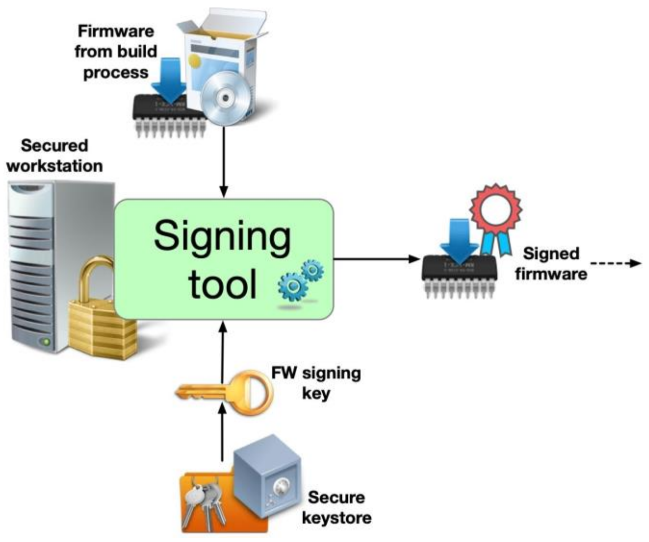
➢ **A-symmetric FW signing**
  - The validating key is stored on the embedded device ("public" key)
  - The signing key is stored on the build environment ("private" key)
    - » Hopefully in a secure manner…

99

99

EMBEDDED USE CASES

FW SIGNING PROCESS



100

100

EMBEDDED USE CASES

A-SYMMETRIC FW SIGNING

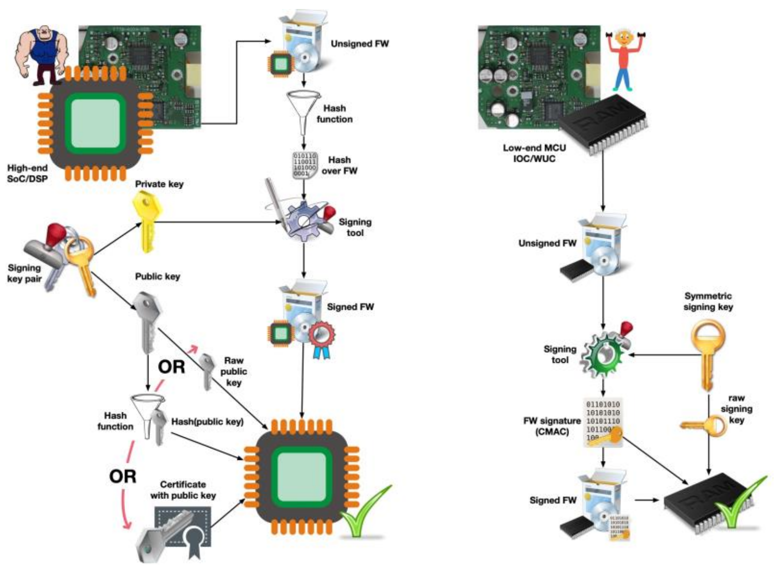
- The FW signing environment is more protected when compared to the ECUs–
  Unless the FW signing environment is connected to the internet…– And NO, a corporate firewall is not enough!
- The ECUs are out there so are much more accessible to adversaries
- When a-symmetric FW signature is used:
  - The signing PRIVATE key does not exist on the embedded device
  - Only the validating PUBLIC key is stored on the embedded device
  - Obtaining the PRIVATE key from the PUBLIC key is too hard
  - Process is secure as long as the PRIVATE key is secure

101

101

EMBEDDED USE CASES

FW SIGNING – TYPICAL ECU



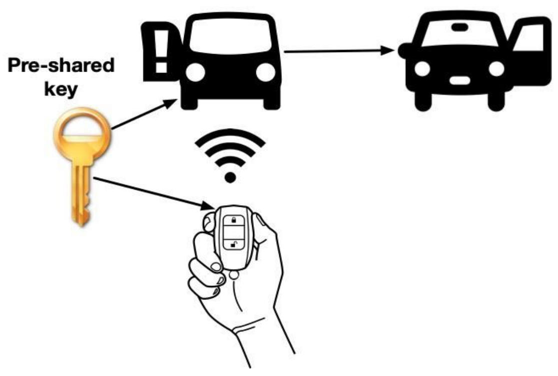
102

102

EMBEDDED USE CASES

ACCESS CONTROL – RKE USE CASE

RKE runs on very low-end platforms →
requires low-footprint cryptography →
symmetric cryptography is used exclusively



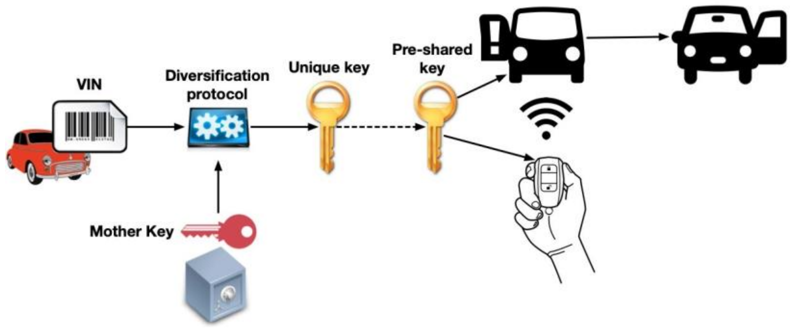
103

103

ANOTHER LESSON FROM THE PAST…

- **The keys that are applied for RKE ~~should~~ must be diversified**
- **A major OEM learned this the hard way, just FOUR keys were used for tens of millions of vehicles**
- **See the paper "*Lock It and Still Lose It - on the (In)Security of Automotive Remote Keyless Entry Systems*" for details***

* Authors:
Flavio D. Garcia & David Oswald, U. of Birmingham; Timo Kasper, Kasper & Oswald GmbH; Pierre Pavlidès, U. of Birmingham

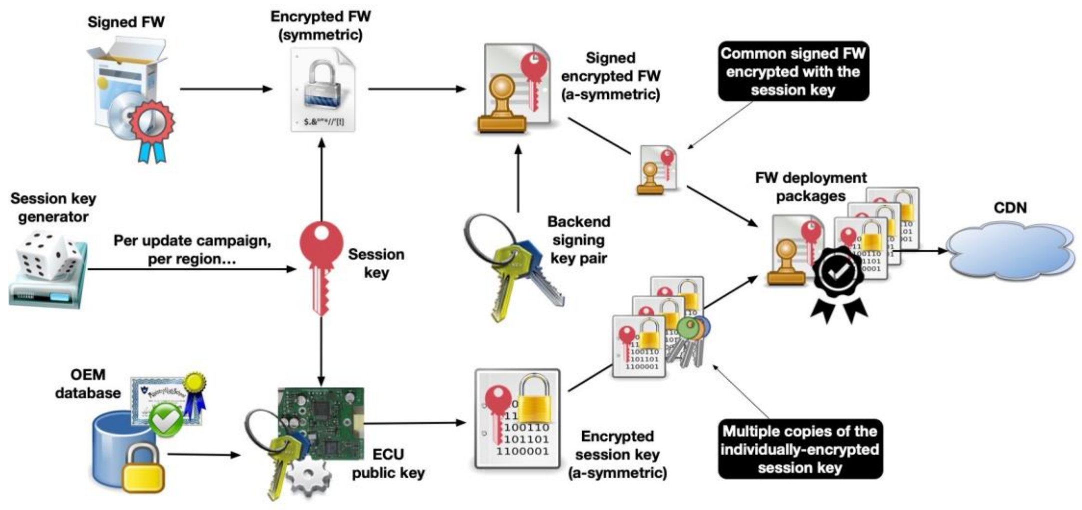


THE OTA USE CASE

- **The OTA process delivers updated FW to the target devices**
- **It usually applies independent signatures and encryption, on top of the regular FW signing and encryption (when applicable)**
- **The size of the payload may be big, so a-symmetric cryptography is not practical**
- **A hybrid method is used:**
- Symmetric session keys are used to encrypt the big OTA payload
- Only the symmetric key is encrypted with the target's individual public key
- Target first decrypts the key and then decrypts the big payload

EMBEDDED USE CASES

THE OTA CASE – AT THE BACKEND

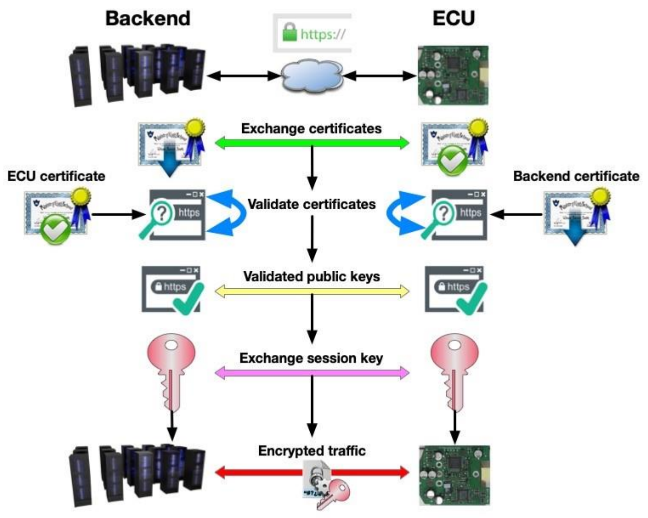

Signed FW
Encrypted FW (symmetric)
Signed encrypted FW (a-symmetric)
Common signed FW encrypted with the session key
Session key generator
Per update campaign, per region…
Session key
Backend signing key pair
FW deployment packages
CDN
OEM database
ECU public key
Encrypted session key (a-symmetric)
Multiple copies of the individually-encrypted session key

106

106

EMBEDDED USE CASES

THE OTA CASE – TRANSPORT LAYER

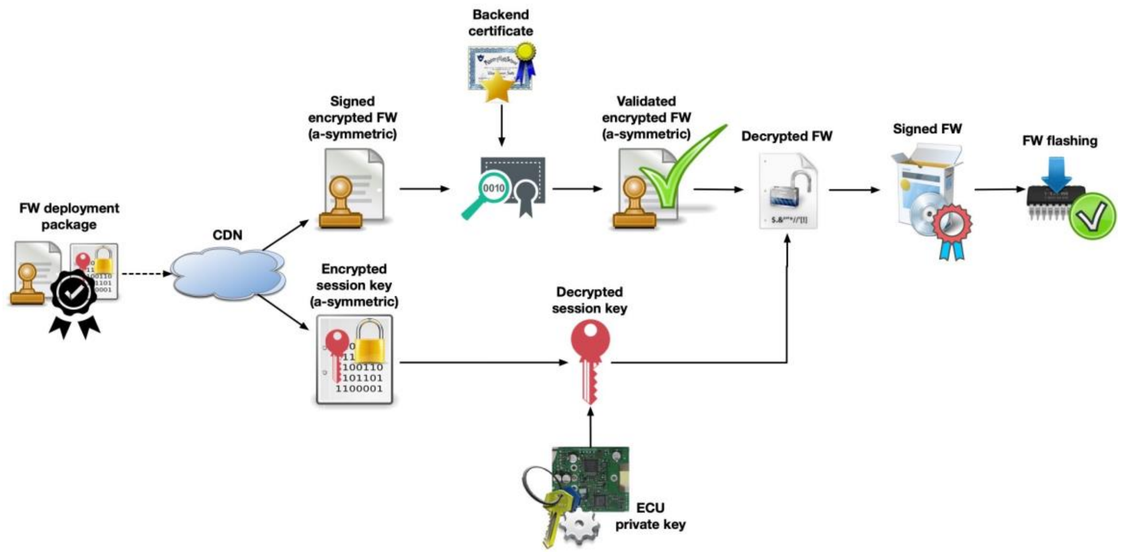


107

107

EMBEDDED USE CASES

THE OTA CASE – AT THE END POINT



108

108

EMBEDDED USE CASES

KEY VALIDATION

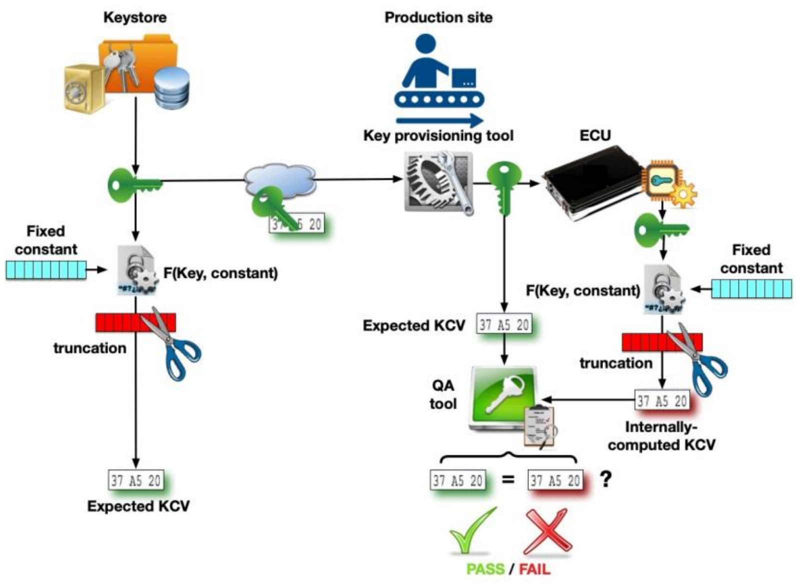
- Manufacturing embedded devices involves a lot of key provisioning
- Production tools upload keys into embedded devices and then need to validate the provisioned keys, to detect corrupted keys and wrong keys
- Direct readback of the keys is very risky so ~~may be~~ must be blocked or impossible
- The alternative method: read a robust checksum that does not disclose the value of the key, but can still validate its correctness/integrity with high assurance
- This is called a "*KCV*" = Key Checksum Value, for symmetric keys
- The term "*thumbprint*" or "*fingerprint*" is used for a-symmetric keys
- KCVs are computed by encrypting a fixed value, defined in the standards, and truncating the result
- Thumbprints are just hashes of the public key, weak hashes may be used

109

109

EMBEDDED USE CASES

KEY VALIDATION PROCESS



110

110

THANK YOU

124