Chapter 3

A Primer in Cryptography

(Crypto means Cryptography, not Cryptocurreny)

Cryptography: Basic terminology

- **plaintext (Klartext)** – original message
- **ciphertext (Chiffrat)** – coded message
- **cipher** / chiffre **(Verschlüsselungsalgorithmus)** – algorithm for transforming plaintext to ciphertext and vice versa
- **key (Schlüssel)** – info used in cipher known only to sender/receiver
- encipher / **encrypt (verschlüsseln)** – converting plaintext to ciphertext – different from **encode** (code without a key)!
- decipher / **decrypt (entschlüsseln)** – recovering plaintext from ciphertext
- **cryptography (Kryptographie)** – study of encryption principles / methods
- **cryptanalysis (Kryptoanalyse)** – study of principles / methods of deciphering ciphertext without knowing key
- **cryptology (Kryptologie)** – scientific field of both cryptography and cryptanalysis

Cryptography: Kerkhoff's principle

„The security of a cryptosystem must not depend on keeping the cryptographic algorithm secret.”

- Security of cipher may only depend on the security of the key
- Always assume all details of the algorithm / method / protocol to be publicly known
- All modern cryptographic methods follow this principle (cf. AES selection process – done completely in the open, with public rounds of discussion)

Cryptography:
Classification of primitives

Remember whole classification (not only for exam)!

- **Cryptographic hash** (0 keys): not reversible

- **Symmetric** (1 **secret** key)
  - symmetric encryption, also called cipher or chiffre
    - block cipher
    - stream cipher
  - symmetric signature, also called message authentication code (MAC)

- **Asymmetric** (2 keys: **public** key and **private** key)
  - key agreement
  - asymmetric encryption
  - asymmetric signature

Cryptography: Classification of primitives

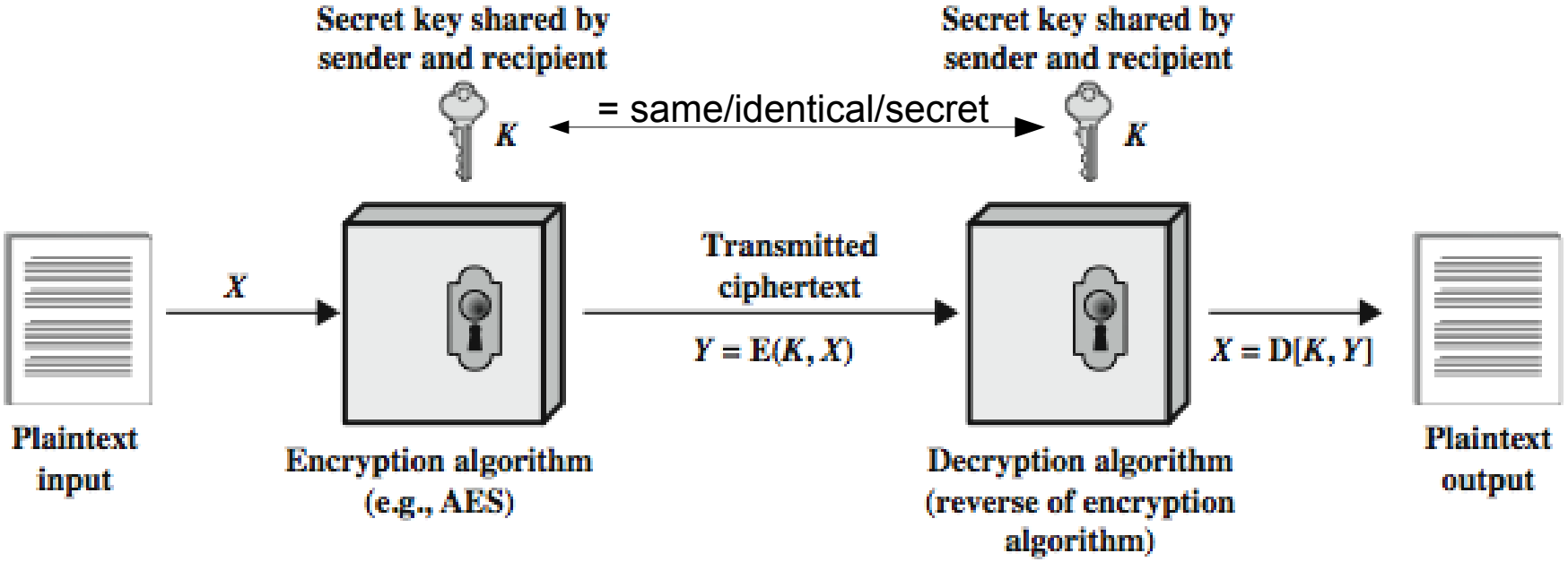
	Symm. cipher	Symm. authen-ticated cipher	Symm. cipher with block tweaks	Crypto-graphic hash	Symm. message authentication code	Key agree-ment	Asymm. encryption	Asymm. signature
Confidentiality	X	X	X				**Careful!**	
Integrity		X		**NO!**	X			with hash
Integrity of data at rest			X					
Authenticity					partial	**NO!**		with public key
Key exchange						X	X	
Non-repudiability								with certificates
Algorithm	AES-CBC AES-CTR ChaCha20	**AES-CCM** ChaCha20 -Poly1305	AES-XTS	SHA-2 **SHA-3**	HMAC-SHA2 **HMAC-SHA3** Poly1305	DH **Curve25519**	**RSA**	RSA **Ed25519**

Cryptography:
Symmetric encryption

- Or conventional / (~~private-key~~) / secret-key / single-key
- Sender and recipient share a common key → must have obtained copies of the secret key in a secure fashion and must keep the key secure
- All classical encryption algorithms are private-key
- Was only type prior to invention of public-key in 1970's
- And by far most widely used

- **the universal technique for providing confidentiality for transmitted or stored data**

Cryptography: Symmetric encryption



Cryptography: Symmetric encryption requirements

- **Two requirements for secure use of symmetric encryption:**
  - a strong encryption algorithm
  - a secret key known only to sender / receiver

- Mathematically have (X=cleartext, Y=ciphertext):
  - Y = E(K, X)
  - X = D(K, Y)
- Assume encryption algorithm is known
- Implies a secure channel to distribute key K

Attacking symmetric encryption

Objective is to recover key, not just message

→ if successful, all future and past messages encrypted with that key are compromised

Cryptanalytic Attacks

- Rely on:
  - nature of the algorithm
  - some knowledge of the general characteristics of the plaintext
  - some sample plaintext-ciphertext pairs
- Exploits the characteristics of the algorithm to attempt to deduce a specific plaintext or the key being used

Brute-Force Attack

- Try all possible keys on some ciphertext until an intelligible translation into plaintext is obtained
- On average half of all possible keys must be tried to achieve success

Cryptanalysis: Attacks

- **brute force:** simply try all possible key combinations

Depending on input knowledge for attack, distinguish between:

- **ciphertext only:** only know algorithm and ciphertext, is statistical, know or can identify/recognize a correct plaintext
- **known plaintext:** know/suspect plaintext and ciphertext
- **chosen plaintext:** select plaintext and obtain ciphertext
- **chosen ciphertext:** select ciphertext and obtain plaintext
- **chosen text:** select plaintext or ciphertext to en/decrypt
- **adaptive chosen (plain-/cipher-)text:** select text based on results of previous tries

Cryptanalysis: Modern methods

- Differential cryptanalysis
  - try to relate differences between plain texts with differences between cipher texts
- Linear cryptanalysis
  - statistical correlations between plain text and cipher text based on structure of cipher are used to estimate key
- Timing (and other so-called **side-channel**) attacks
  - measuring CPU time taken for different operations during the execution of a cipher
  - when CPU operations are dependent on data (e.g. plain text and/or key), they might take different execution time
  - statistical analysis concerning probability of key and/or plain text combinations
  - given sufficient input data (e.g. number of operations with the same key but different plain texts), can estimate key (and/or plain text)

Cryptanalysis: Definitions

- **Unconditional security**
  - no matter how much computer power or time is available, the cipher cannot be broken since the ciphertext provides insufficient information to uniquely determine the corresponding plaintext
  - sometimes called “Shannon unconditional security” after the seminal paper “Communication Theory of Secrecy Systems” by Claude Elwood Shannon, 1949

- **Computational security**
  - given limited computing resources (e.g. time needed for calculations is greater than age of universe), the cipher cannot be broken

- **“Acceptable“ security**
  - given **assumptions** on the possibilities of attackers (computing power available, budget, time-constraints...), the cipher cannot be broken

Cryptanalysis:
Brute force search

- Always possible to simply try every key
- Most basic attack, proportional to key size
- Assume either to know or to recognize plaintext
- Note concerning numbers: it will only get faster!
- E.g. 2010 Intel AES-NI supported ca. 50 Mio. AES blocks/s on each core

Key Size (bits)	Number of Alternative Keys	Time Required at 1 Decryption/μs	Time Required at 10^6 Decryptions/μs
32	$2^{32} = 4.3 \times 10^9$	$2^{31}\ \mu s$ = 35.8 minutes	2.15 milliseconds
56	$2^{56} = 7.2 \times 10^{16}$	$2^{55}\ \mu s$ = 1142 years	10.01 hours
128	$2^{128} = 3.4 \times 10^{38}$	$2^{127}\ \mu s$ = 5.4×10^{24} years	5.4×10^{18} years
168	$2^{168} = 3.7 \times 10^{50}$	$2^{167}\ \mu s$ = 5.9×10^{36} years	5.9×10^{30} years
26 characters (permutation)	$26! = 4 \times 10^{26}$	$2 \times 10^{26}\ \mu s = 6.4 \times 10^{12}$ years	6.4×10^6 years

Symmetric encryption:
One-Time Pad (OTP)

- If a **truly random key as long as the message** is used, the cipher will be **unconditionally** secure
- Called a **One-Time pad**
- Is unbreakable since ciphertext bears no statistical relationship to the plaintext
  - **This is the only cipher that is provably secure under Shannon unconditional security!**
  - since for any plaintext and any ciphertext there exists a key mapping one to other
- Can **only use the key once** though
- Problems in generation and safe distribution of key

- **Summary of requirements for One-Time pad (definition):**
  - key is (at least) as long as the message
  - key is generated by truly random source
    (no statistically significant patterns and unpredictable by attackers)
  - key is only used once

Remember!

Symmetric encryption:
Block vs. stream ciphers

Block ciphers

- Block ciphers process messages in blocks, each of which is then en-/decrypted
- Produces an output block for each input block
- Like a substitution on very big characters
  - 64 bits or more, today use at least 128
- Can reuse keys – but only if used with suitable block cipher mode

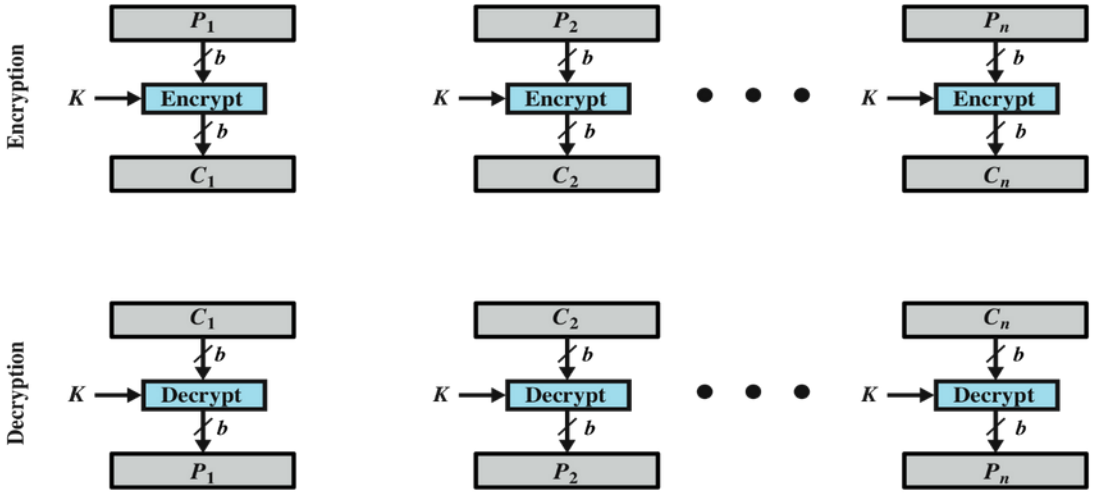
Stream ciphers

- Stream ciphers process messages continuously a bit or byte at a time when en/decrypting by combining input with pseudorandom “key”-stream
- Pseudorandom stream is one that is unpredictable without knowledge of the input key
- Produces output one element at a time
- Primary advantages are that they don't need padding and are in many cases faster and use far less code

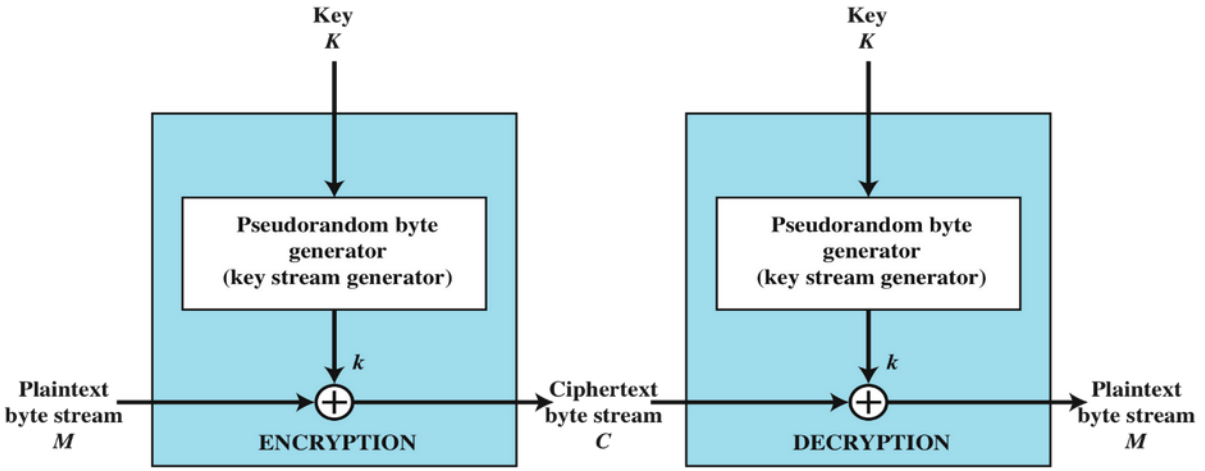
Many current ciphers are block ciphers

- Better analyzed, broader range of applications
- But: as of 2014, renewed interest in stream ciphers, see e.g. current ChaCha20 use as a partial result of eSTREAM project by EU ECRYPT network to "identify new stream ciphers suitable for widespread adoption”

Symmetric encryption: Block vs. stream ciphers



(a) Block cipher encryption (electronic codebook mode)



(b) Stream encryption

Symmetric encryption: Block cipher principles

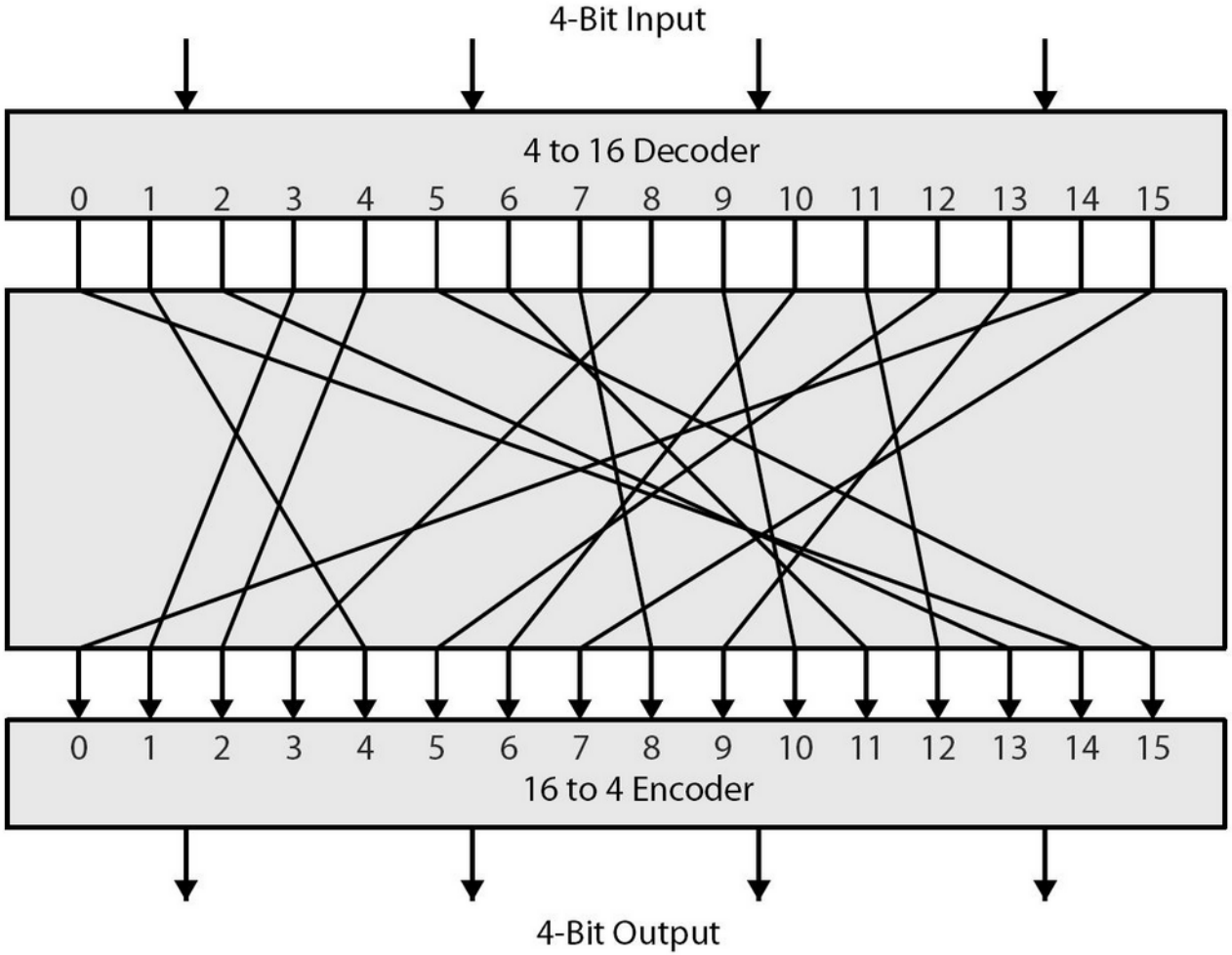
- Block ciphers look like an extremely large substitution
- Ideal block cipher, e.g. with 128 bits block size:
  - en-/decryption is a mapping function *e:* $2^{128} \rightarrow 2^{128}$
  - "key" is a table of 2^{128} entries with 128 bits length for each entry (mapping each of the possible 2^{128} blocks to another block)
  - Side note: assume 10^{78} to 10^{82} atoms in the known, observable universe [1] (very roughly around 2^{256}) → seems hard to store single key of 128 x 2^{128} bits
  - key space is $(2^{128})!$

This means factorial, as in "I tell you, 230 - 220 x 0.5 = 5!"

- Instead create from smaller building blocks
  - very often use keys in the range of the block size (e.g. AES is defined with 128 bits block size and supports 128, 192, or 256 bits key length)
  - these keys only allow a smaller key space than ideal block cipher, but block size becomes limiting factor for statistical attacks if key is much longer (cf. 3DES)
- Using idea of a product cipher (i.e. combined substitution and permutation)
- Most symmetric block ciphers are based on a **Feistel Cipher Structure**

[1] https://www.universetoday.com/36302/atoms-in-the-universe/

Symmetric encryption: Ideal block cipher



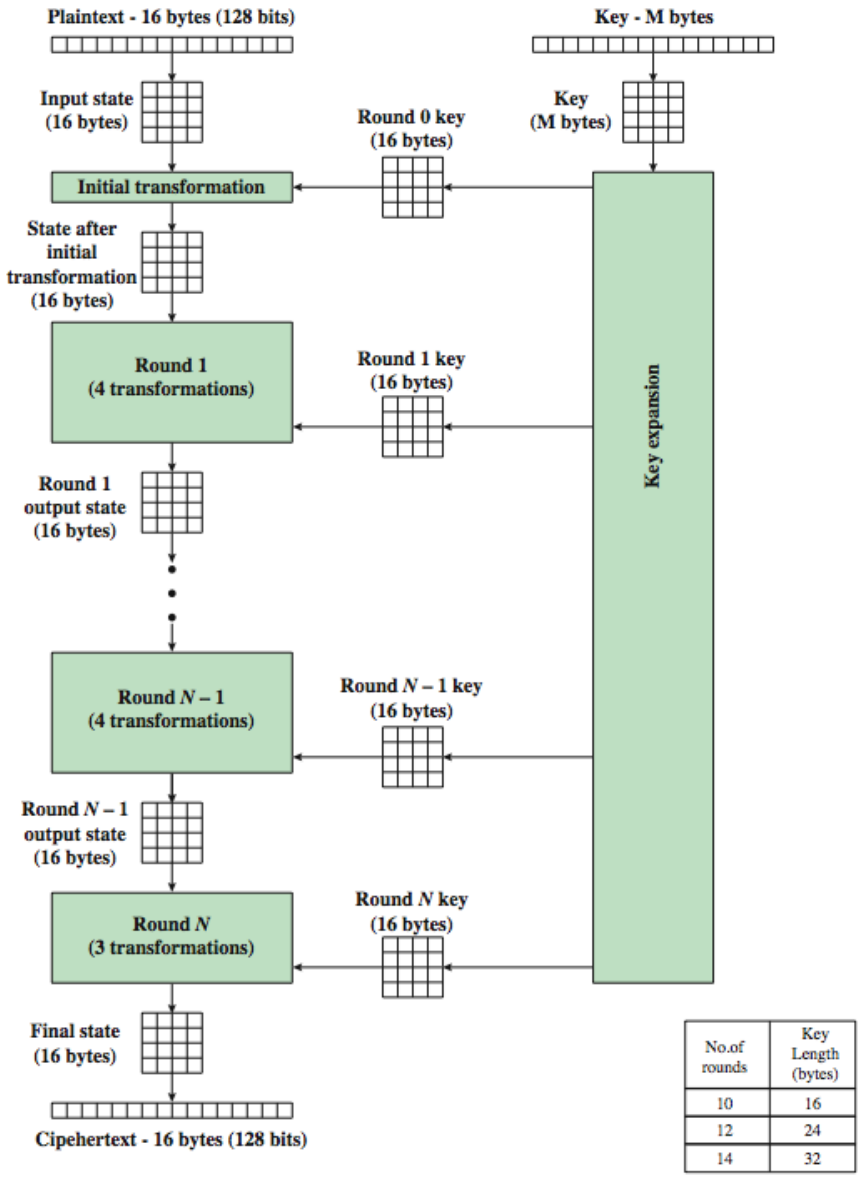
Advanced Encryption Standard (AES)

- (Long, long ago) it became clear a replacement for DES (Data Encryption Standard, used for decades) was needed
  - have theoretical attacks that can break it
  - have demonstrated exhaustive key search attacks
  - can use Triple-DES – but slow, **has small blocks**
- Process for AES was open competition (first in that form)
  - US NIST issued call for ciphers in 1997
  - 15 candidates accepted in June 1998
  - 5 were shortlisted in August 1999
  - Rijndael was selected as the AES in Oct-2000
  - issued as FIPS PUB 197 standard in Nov-2001

AES cipher - Rijndael

- Designed by *Rijmen-Daemen* in Belgium
- Has 128/192/256 bit keys, 128 bit block length
  - original Rijndael specification allows 128-256 bit block length in 32 bit increments
- An **iterative** rather than **Feistel** cipher
  - processes data as block of 4 columns of 4 bytes
  - operates on entire data block in every round
- Designed to be:
  - resistant against known attacks
  - speed and code compactness on many CPUs
  - design simplicity

AES: Encryption



No.of rounds	Key Length (bytes)
10	16
12	24
14	32

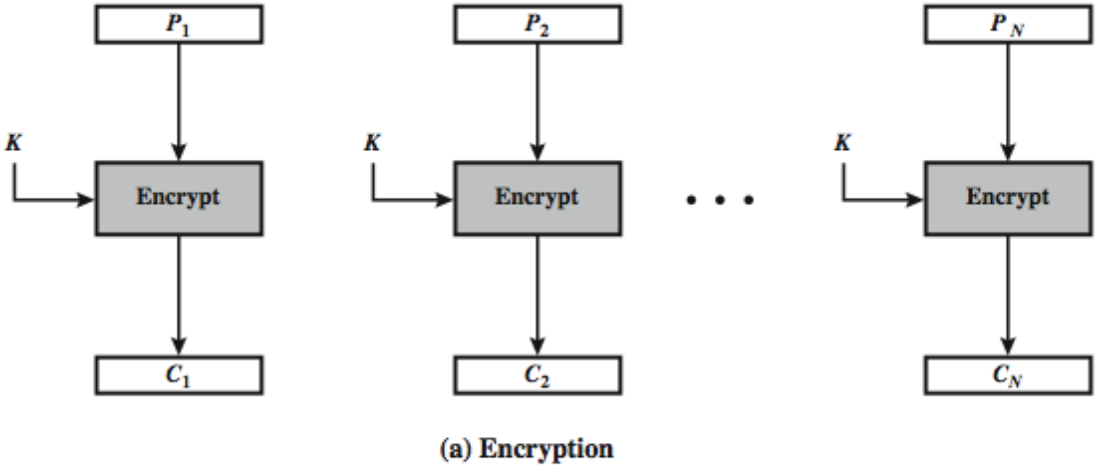
Modes of operation

- Block ciphers encrypt fixed size blocks
  - e.g. AES encrypts 128-bit blocks
- Need some way to en/decrypt arbitrary amounts of data in practice
- NIST SP 800-38A defines 5 **modes**
- Have **block** and **stream** modes
- To cover a wide variety of applications
- *Can be used with **any** block cipher*

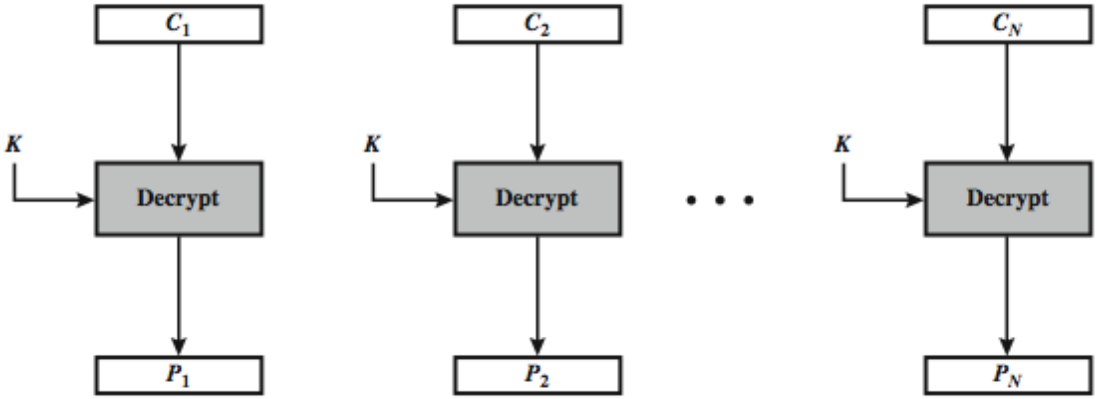
Block cipher modes:
Electronic Code Book (ECB)

- Message is broken into independent blocks which are encrypted
- Each block is a value which is substituted, like a codebook, hence name
- Each block is encoded independently of the other blocks
  $C_i = E_K(P_i)$
- Uses: secure transmission of single values

Electronic Code Book (ECB)



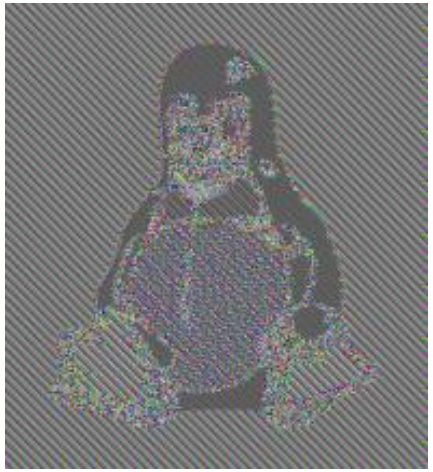
(a) Encryption



(b) Decryption

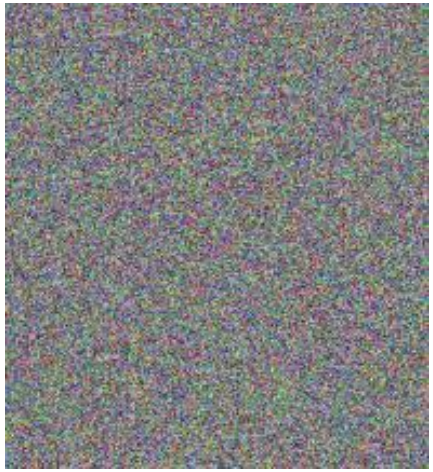
Block cipher modes: Advantages/Limitations of ECB

- Message repetitions may show in ciphertext
  - if aligned with message block
  - particularly with data such as graphics
  - or with messages that change very little, which become a code-book analysis problem
  - one message broken → this message “stays” broken (repetitions!)
- Weakness is due to the encrypted message blocks being independent
- Main use is sending a few blocks of data

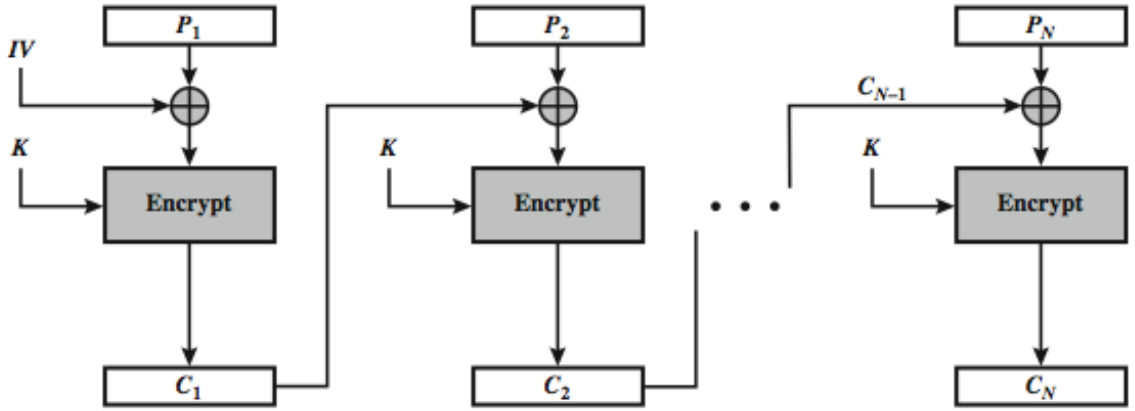


Block cipher modes:
Cipher Block Chaining (CBC)

- Message is broken into blocks
- Linked together in encryption operation
- Each previous cipher block is chained with current plaintext block, hence name
- Use Initialization Vector (IV) to start process ⇒ need to transmit IV
  $C_i = E_K(P_i \text{ XOR } C_{i-1})$
  $C_0 = E_K(IV)$
- Uses: bulk data encryption, authentication in the form of CBC-MAC

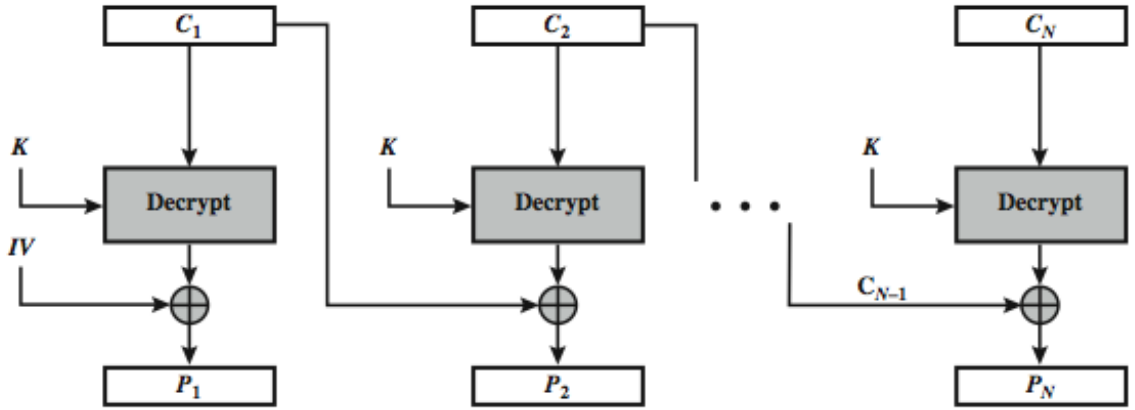


Cipher Block Chaining (CBC)



(a) Encryption

Careful: Changing one bit in C_1 will "destroy" all of P_1, and flip exactly the matching Bit in P_2



(b) Decryption

Block cipher modes:
Message padding

- At end of message must handle a possible last short block
  - which is not as large as blocksize of cipher
  - pad either with known **non-data** value (e.g. nulls)
  - or pad last block along with count of pad size
    - e.g. [b1 b2 b3 0 0 0 0 5]
    - means to have 3 data bytes, then 5 bytes pad+count
  - this may require an extra entire block over those in message
    - message ends with …, 0 0 3, 0 2, 1 → How to distinguish from a short block?
- There are other, more esoteric modes, which avoid the need for an extra block

Block cipher modes: Advantages/Limitations of CBC

- A ciphertext block depends on **all** blocks before it
- Any change to a block affects all following ciphertext blocks

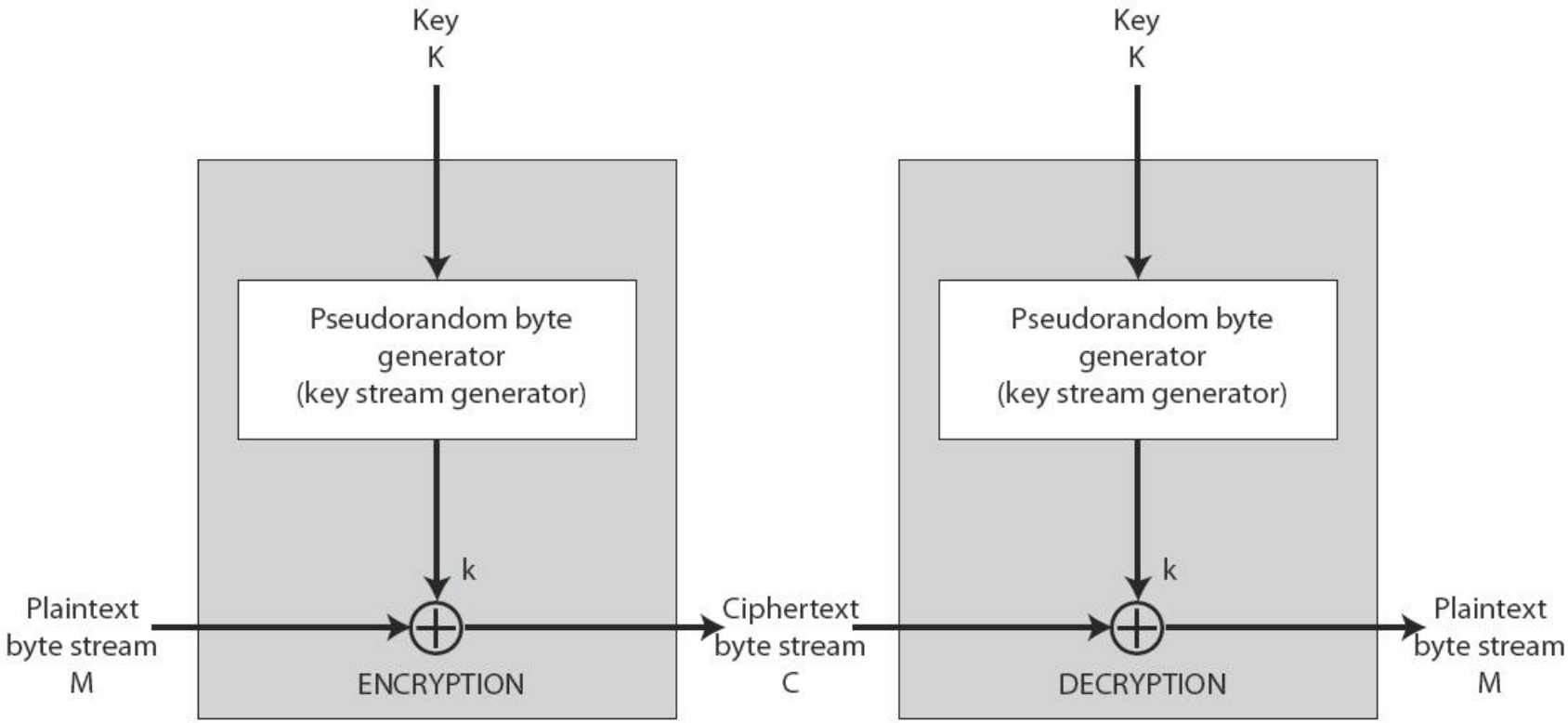
Problems

- Issues with padding in MAC-then-encrypt use especially in TLS (see 2013 TLS attacks)
  - check e.g. https://www.youtube.com/watch?v=ifVD8BqNONk for padding oracle attacks ["Scalable Scanning and Automatic Classification of TLS Padding Oracle Vulnerabilities", Usenix Security 2019]
- Need **Initialization Vector** (IV)
  - which must be known to sender and receiver
  - if sent in clear, attacker can change bits of first block, and change IV to compensate
  - hence IV must either be a fixed value (as in EFTPOS)
    - same cleartext with same key → same ciphertext...
  - or must be sent encrypted in ECB mode before rest of message

Stream modes of operation

- Block modes encrypt entire block
- May need to operate on smaller units
  - real time data
- Stream modes **convert block cipher into stream cipher**
  - cipher feedback (CFB) mode
  - output feedback (OFB) mode
  - counter (CTR) mode
- Use block cipher as some form of **pseudo-random number** generator

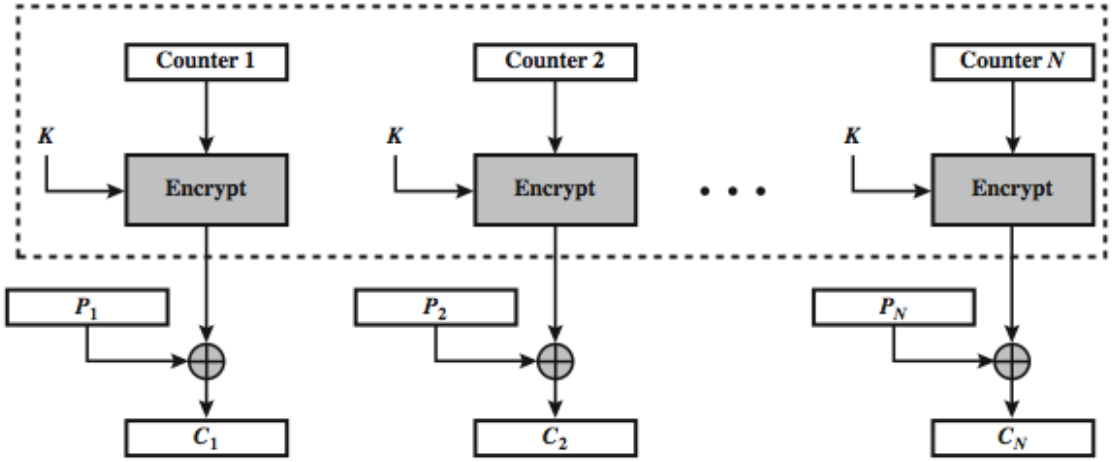
Stream cipher structure



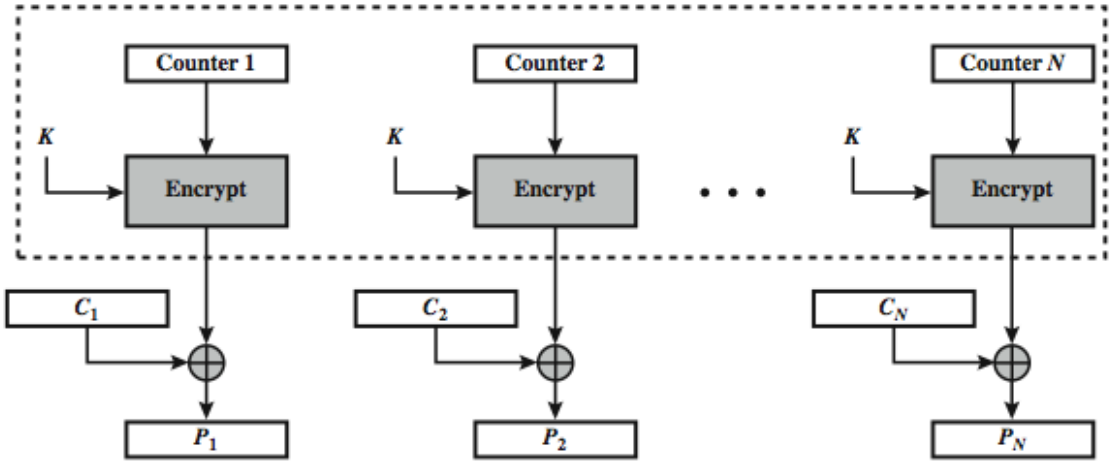
Block cipher modes:
Counter (CTR)

- A “new” mode, though proposed early on
- Similar to OFB but encrypts counter value rather than any feedback value
- Must have a different key and counter value for every plaintext block (**never reused**)
  $O_i = E_K(i)$
  $C_i = P_i \text{ XOR } O_i$
- Uses: high-speed network encryption, encrypting data for random access

Counter (CTR)



(a) Encryption



(b) Decryption

Block cipher modes: Advantages/Limitations of CTR

- Efficiency
  - can do parallel encryptions in hardware or software
  - can preprocess in advance of need
  - good for bursty high speed links
- Random access to encrypted data blocks
- Provable security (as good as other modes)
- But must ensure never to reuse key/counter values, otherwise could break

Block cipher modes: XTS-AES

- New mode, for block oriented storage use
  - in IEEE Std 1619-2007
- Concept of tweakable block cipher
- Different requirements to transmitted data
- Uses AES twice for each block
  $T_j = E_{K2}(i)$ XOR α^j
  $C_j = E_{K1}(P_j$ XOR $T_j)$ XOR T_j
  where i is tweak (sector number) and j is block offset in sector
  α is a special polynom (Galois field multiplication)
- Each sector may have multiple blocks
- (At least) 2 AES en-/decryption operations per block

Block cipher modes: Advantages/Limitations of XTS

- Efficiency
  - can do parallel encryptions in hardware or software
  - random access to encrypted data blocks
- Has both nonce and counter
- Addresses security concerns related to stored data
- **No authentication of data**
- Complications if sector size is not multiple of block size

Authenticated encryption Block cipher modes: **Counter with CBC-MAC (CCM)**

- CCM mode combines the well-known counter (CTR) mode of encryption with the well-known CBC-MAC mode of authentication
  - variation of encrypt-and-MAC approach (see later for others)
- Allows to use **same block cipher with same key** for ensuring confidentiality **and** authenticity/integrity
  - all previous modes only provide confidentiality and need additional MAC (Message Authentication Code) or digital signature to provide authenticity/integrity
- Only requires encryption to be implemented, no decryption function
  - CCM currently only defined for block ciphers with 128 bit block size
  - RFC 3610 defines AES-CCM
  - designed by Russ Housley, Doug Whiting and Niels Ferguson
- Currently used in wireless network standards
  - IEEE 802.11i (WiFi WPA2 with CCMP), e.g. NIST SP 800-38C
  - ZigBee
  - RFC 4309 defines use of AES-CCM for IPsec (not yet in widespread use)
- Has been criticized for **not being online** and for being complex
  - see [Rogaway and Wagner 2003: “A Critique of CCM”]

Authenticated encryption Block cipher modes: **Galois Counter Mode (GCM)**

- Fast, online, not patented
- Standardized for TLS, IPsec, and others
- Implementation is difficult, but standard implementations widely available (e.g. OpenSSL)
  - Intel AES-NI hardware instructions provide speed-up
- Security is problematic with short MAC tags
  - TLS and IPsec define only 96 bits
  - see e.g. https://eprint.iacr.org/2016/475.pdf
  - easy to get implementation wrong, with potentially disastrous failure of message authentication property when nonces are re-used: http://arstechnica.com/security/2016/05/faulty-https-settings-leave-dozens-of-visa-sites-vulnerable-to-forgery-attacks/
- Avoid implementing it yourself!
  - if not completely sure about the implementation, avoid the mode

Authenticated encryption Block cipher modes: **Offset Codebook Mode (OCB)**

- **Fast**, online, **patented**
- Technically one of the best modes
  - https://blog.cryptographyengineering.com/2012/05/19/how-to-choose-authenticated-encryption/
- Patent recently free to use for open source
  - http://web.cs.ucdavis.edu/~rogaway/ocb/license.htm
- Some of the patents expired in April 2016
  - https://pthree.org/2016/03/31/two-ocb-block-cipher-mode-patents-expired-due-to-nonpayment/

RC4

- A proprietary cipher owned by RSA DSI designed by Ron Rivest
- Variable key size, byte-oriented **stream cipher**
- Previously widely used (older SSL/TLS, wireless WEP / WPA with TKIP)

Executive summary: don’t use anymore. Really.

RC4 security

- Some doubt for years, but only recently broken
  - [Nadhem AlFardan, Dan Bernstein, Kenny Paterson, Bertram Poettering, Jacob Schuldt: “On the Security of RC4 in TLS and WPA” and “Biases in the RC4 keystream” (presentation at http://www.isg.rhul.ac.uk/tls/), Usenix 2013]
  - result is very non-linear
- Since RC4 is a stream cipher, must **never reuse a key**
- Have a concern with WEP, but due to key handling rather than RC4 itself
- Standard use in TLS now broken (see 2013 paper cited above)

  → **don't use RC4 anymore!**
- Example of newer stream cipher: **ChaCha20** (variant of Salsa20), specified in RFC7539 (https://tools.ietf.org/html/rfc7539)

Public-key cryptography

- Probably most significant advance in the 3000 year history of cryptography
- Uses **two** keys in the form of a **keypair** – a public and a private key
- **Asymmetric** since parties are **not** equal
- Uses clever application of number theoretic concepts to function
- Complements rather than replaces symmetric key cryptography

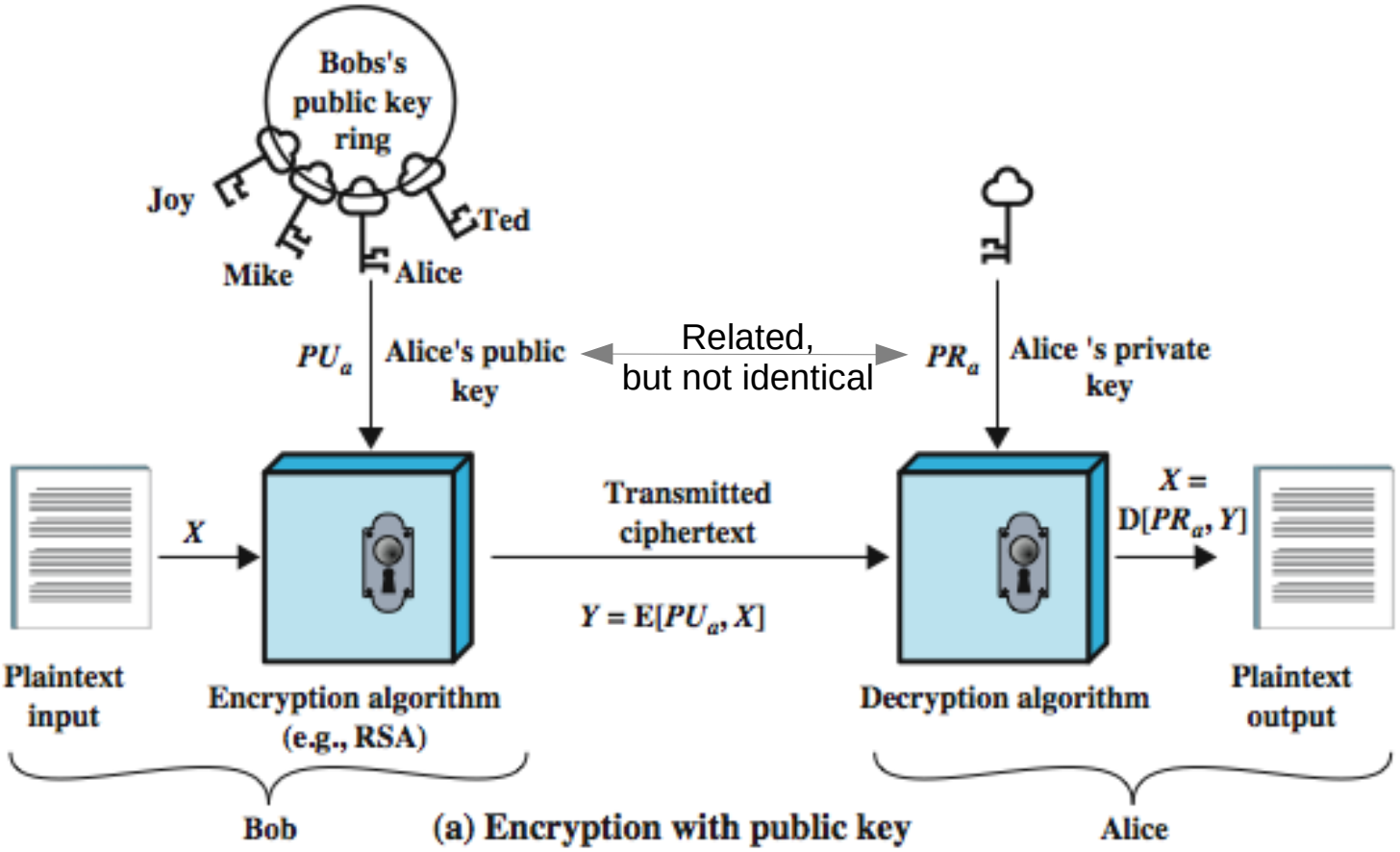
Why public-key cryptography

- Developed to address two key issues:
  - **key distribution** – how to have secure communications in general without having to trust a KDC (key distribution center) with your symmetric/secret key
  - **digital signatures** – how to verify a message comes intact from the claimed sender
- Public invention due to Whitfield Diffie & Martin Hellman at Stanford University in 1976 (article “New direction in cryptography”)
  - known earlier in classified community

Public-key cryptography

- **Public-key/two-key/asymmetric** cryptography involves the use of **two** keys:
  - a **public key**, which may be known by anybody, and can be used to **encrypt messages**, and **verify signatures**
  - a related **private key**, known only to the recipient, used to **decrypt messages**, and **sign** (create) **signatures**
- **Infeasible to determine private key from public**
  - Note: The reverse is typically easy
- **Infeasible to decrypt message or sign without knowing private key**
- Is **asymmetric** because
  - those who **en**crypt messages or **verify** signatures **cannot** **de**crypt messages or **create** signatures

Public-key cryptography



Symmetric (secret/single-key) vs. asymmetric (public-key)

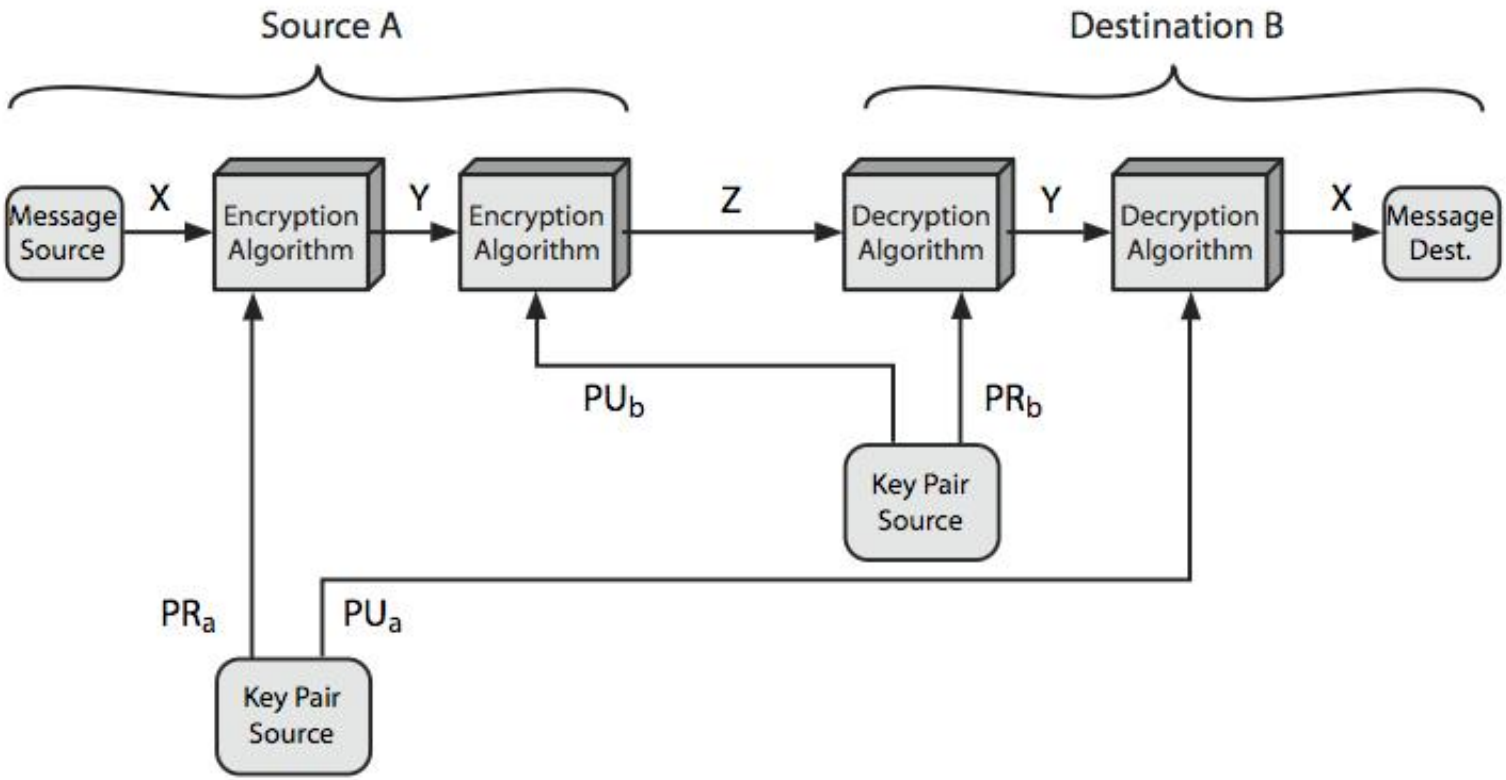
Symmetric encryption

- Needed to work
  - same algorithm **with same key**
  - sender and receiver **share key**
- Needed for security
  - **single key** must be kept secret
  - knowledge of algorithm + samples of cipher-/plaintext must be insufficient to determine this secret key

Asymmetric encryption

- Needed to work
  - same algorithm **with pair of keys** (one to encrypt, one to decrypt)
  - sender and receiver **each have a pair of keys**
- Needed for security
  - **private part of keypair** must be kept secret
  - knowledge of algorithm + public part of keypair + samples of cipher-/plaintext must be insufficient to determine private key

Public-key cryptosystems



Public-key applications

- Can classify uses into 3 categories:
  - **encryption/decryption** (provide confidentiality/secrecy)
  - **digital signatures** (provide authentication)
  - **key exchange** (of session keys)
- Some algorithms are suitable for all uses (e.g. RSA), others are specific to one (e.g. Diffie-Hellman only for key exchange, different elliptic curve based algorithms for different purposes)

Public-key requirements

- Public-key algorithms rely on two keys where:
  - it is computationally infeasible to find decryption key knowing only algorithm and encryption key
  - it is computationally infeasible to en-/decrypt messages when the relevant (en-/decrypt) key is not known
  - it is computationally easy to en-/decrypt messages when the relevant (en-/decrypt) key is known
  - it is computationally easy to generate keypair
  - especially useful if either of the two related keys can be used for encryption, with the other used for decryption (for some algorithms)
- These are formidable requirements which only a few algorithms have satisfied

Public-key requirements

- Need a trapdoor one-way function
- One-way function has
  - Y = f(X) easy
  - X = $f^{-1}(Y)$ infeasible
- A trap-door one-way function has
  - Y = $f_k(X)$ easy, if k and X are known
  - X = $f_k^{-1}(Y)$ easy, if k and Y are known
  - X = $f_k^{-1}(Y)$ infeasible, if Y known but k not known
- A practical public-key scheme depends on a suitable trap-door one-way function

Security of public-key schemes

- Like private key schemes brute force **exhaustive search** attack is always theoretically possible
- But keys used are too large (>= 2048 bits for classical, >= 256 bits for elliptic curve variants)
- Security relies on a **large enough** difference in difficulty between **easy** (en-/decrypt) and **hard** (cryptanalysis) problems
- More generally the **hard** problem is known, but is made hard enough to be impractical to break
- Requires the use of **very large numbers**
- Hence is **slow** compared to private key schemes

RSA

- By Rivest, Shamir & Adleman of MIT in 1977
- Best known and widely used public-key scheme
- Based on exponentiation in a finite (Galois) field over integers modulo a prime
  - Note: exponentiation takes $O((\log n)^3)$ operations (easy)
- Uses large integers (e.g. 2048 bits)
- Security due to cost of factoring large numbers
  - Note: factorization takes $O(e^{\log n \log \log n})$ operations (hard)

RSA key generation

- Users of RSA must:
  - determine two primes at random - p, q
  - calculate n = p * q and φ=(p-1)*(q-1)
  - select either e or d (with special relation to φ) and compute the other
    - e*d mod φ = 1
- Primes p,q must not be easily derived from modulus n=p*q
  - must be sufficiently large
  - typically guess and use probabilistic test whether a prime
    - if its not a prime and still passed the test → unlucky & insecure
- Exponents e, d are inverses, so use inverse algorithm to compute the other

RSA security

- Possible approaches to attacking RSA are:
  - brute force key search - infeasible given size of numbers
  - mathematical attacks - based on difficulty of computing φ(n), by factoring modulus n (hard without a quantum computer with sufficiently many qbits...)
  - timing attacks - on running of decryption
  - chosen ciphertext attacks - given properties of RSA

Factoring problem

- Mathematical approach takes 3 forms:
  - factor n=p*q, hence compute φ(n) and then d
  - determine φ(n) directly and compute d
  - find d directly
- Currently believe all equivalent to factoring
  - have seen slow improvements over the years
    - see e.g. https://en.wikipedia.org/wiki/RSA_Factoring_Challenge for challenge (cash prices only active until 2007, but factoring still ongoing)
  - biggest improvement comes from improved algorithm
    - cf. QS to GHFS to LS
  - currently assume >2048 bit RSA is secure, but don't use less than 3072 for new use cases
    - ensure p, q of similar size and matching other constraints
  - known to be computable efficiently with quantum computers (as soon as they reach required qbit register size)

RSA number	Decimal digits	Binary digits	Cash prize offered	Factored on
RSA-100	100	330	US$1,000	April 1, 1991[5]
RSA-110	110	364	US$4,429	April 14, 1992[5]
RSA-120	120	397	US$5,898	July 9, 1993[6]
RSA-129	129	426	US$100	April 26, 1994[5]
RSA-130	130	430	US$14,527	April 10, 1996
RSA-140	140	463	US$17,226	February 2, 1999
RSA-150	150	496		April 16, 2004
RSA-155	155	512	US$9,383	August 22, 1999
RSA-160	160	530		April 1, 2003
RSA-170	170	563		December 29, 2009
RSA-576	174	576	US$10,000	December 3, 2003
RSA-180	180	596		May 8, 2010
RSA-190	190	629		November 8, 2010
RSA-640	193	640	US$20,000	November 2, 2005
RSA-200	200	663		May 9, 2005
RSA-210	210	696		September 26, 2013[8]
RSA-704	212	704	US$30,000	July 2, 2012
RSA-220	220	729		May 13, 2016
RSA-230	230	762		August 15, 2018
RSA-232	232	768		February 17, 2020[9]
RSA-768	232	768	US$50,000	December 12, 2009
RSA-240	240	795		Dec 2, 2019[10]
RSA-250	250	829		Feb 28, 2020[11]

Timing attacks

- Developed by Paul Kocher in mid-1990's
- Exploit timing variations in operations
  - e.g. multiplying by small vs large number
  - or IF's varying which instructions executed
- Infer operand size based on time taken
- RSA exploits time taken in exponentiation
- Countermeasures
  - use constant exponentiation time
  - add random delays
  - blind values used in calculations

Chosen ciphertext attack

- RSA is vulnerable to a Chosen Ciphertext Attack (CCA)
- Attacker chooses ciphertexts and gets decrypted plaintext back
- Choose ciphertext to exploit properties of RSA to provide info to help cryptanalysis
- Can counter with random pad of plaintext
- Or best: **use Optimal Asymmetric Encryption Padding (OASP)**

Diffie-Hellman key exchange (DH)

- First public-key type scheme proposed
- By Diffie & Hellman in 1976 along with the exposition of public key concepts
  - note: now know that Williamson (UK CESG) secretly proposed the concept in 1970
  - Ralph Merkle developed similar method independently, but published only slightly later
    - In 2002, Hellman suggested the algorithm be called Diffie–Hellman–Merkle key exchange in recognition of Ralph Merkle's contribution to the invention of public-key cryptography (Hellman, 2002).
- Is a practical method for public exchange of a secret key
- Used widely (in classical variant based on exponentiation in finite field or more recently in Elliptic Curve variants)

Diffie-Hellman key exchange (DH)

- A public-key distribution scheme
  - cannot be used to exchange an arbitrary message
  - rather it can establish a common key
  - known only to the two participants (when only passive attacks are assumed)

Remember!

- Value of key depends on the participants (and their private and public key information)
- Based on exponentiation in a finite (Galois) field (modulo a prime or a polynomial) – easy
- Security relies on the difficulty of computing discrete logarithms (similar to factoring) – hard (without quantum computers)

Diffie-Hellman setup

- All users agree on global parameters:
  - large prime integer or polynomial q
  - a being a primitive root $\bmod\ q$
- Each user (e.g. A) generates their key
  - chooses a secret key (number): $x_A < q$
  - compute their **public key**: $y_A = a^{x_A} \bmod q$
- Each user makes public that key y_A
  - e.g. transmission to the communication partner in cleartext

Diffie-Hellman key exchange

- Shared session key for users A and B is K_{AB}:

  $K_{AB} = a^{x_A \cdot x_B} \bmod q$

  $= y_A^{x_B} \bmod q$ (which **B** can compute)

  $= y_B^{x_A} \bmod q$ (which **A** can compute)

- K_{AB} is used as session key in private-key encryption scheme between Alice and Bob
- If Alice and Bob subsequently communicate, they will have the **same** key as before, unless they choose new public-keys
- Attacker needs an x, must solve discrete log

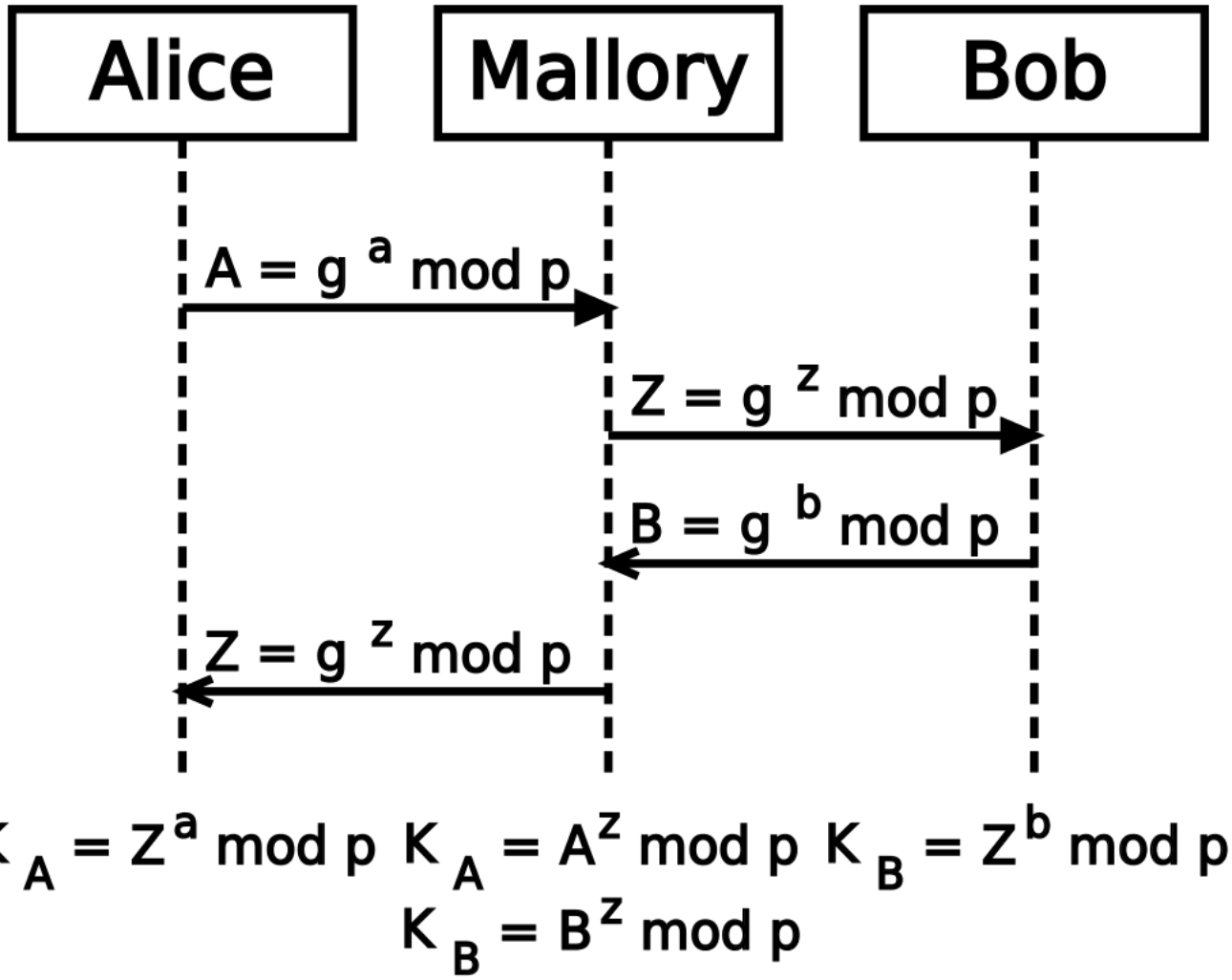
Remember!

On-path attack (OPA) (aka Man-in-the-Middle (MITM) attack)

1. Mallory prepares attack by creating two private / public keys
2. Alice transmits her public key to Bob
3. Mallory intercepts this and transmits his first public key to Bob. Mallory also calculates a shared key with Alice
4. Bob receives the public key and calculates the shared key (with Mallory instead of Alice)
5. Bob transmits his public key to Alice
6. Mallory intercepts this and transmits his second public key to Alice. Mallory calculates a shared key with Bob
7. Alice receives the key and calculates the shared key (with Mallory instead of Bob)
8. Mallory can then intercept, decrypt, re-encrypt, forward all messages between Alice and Bob

Remember!

On-path attack



Source: https://commons.wikimedia.org/wiki/File:Man-in-the-middle_attack_of_Diffie-Hellman_key_agreement.svg

Elliptic Curve Cryptography (ECC)

- Majority of public-key crypto (RSA, DH) use either integer or polynomial arithmetic with very large numbers/polynomials
- Imposes a significant load in storing and processing keys and messages
- An alternative is to use elliptic curves
- Offers same security with smaller bit sizes

Comparable key sizes for equivalent security

Symmetric scheme (key size in bits)	ECC-based scheme (size of *n* in bits)	RSA/DSA (modulus size in bits)
56	112	512
80	160	1024
112	224	2048
128	256	3072
192	384	7680
256	512	15360

Zero knowledge proofs

- Sometimes would like to prove knowledge of a secret without revealing anything about that secret – including the identity of the prover (signer)
- Example 1: “prove that you know a password” → “password is X”
  - if verifier is malicious (or broken), can leak the secret
- Example 2: signing petition by proving to be member of a group (e.g. citizen of a country)
  - need to remain anonymous within that group
  - but standard asymmetric signatures reveal signer
    - good if non-repudiability is desired (legal signatures)
    - bad for privacy
- Details
  - https://blog.cryptographyengineering.com/2014/11/27/zero-knowledge-proofs-illustrated-primer/
  - https://blog.cryptographyengineering.com/2017/01/21/zero-knowledge-proofs-an-illustrated-primer-part-2/
  - https://zkproof.org/2020/08/12/information-theoretic-proof-systems/
  - https://medium.com/witnet/spartan-zksnarks-without-trusted-setup-d117ded96e6f

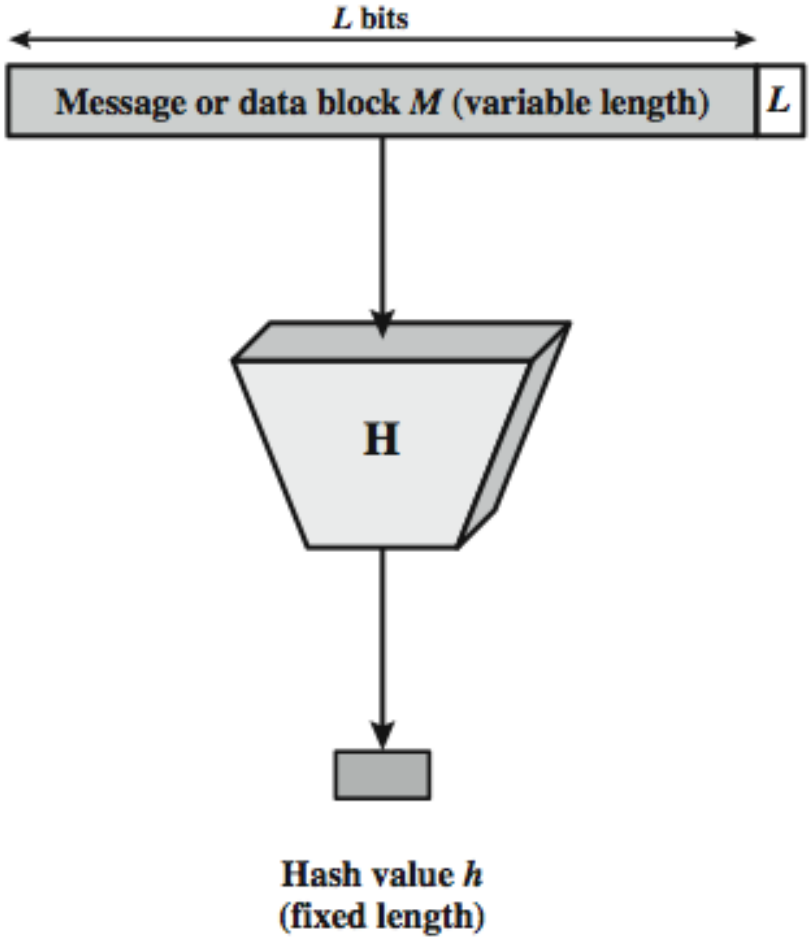
(Cryptographic) **Hash functions**

- Condenses arbitrary message to fixed size
  h = H(M)
- Hash used to detect changes to message
- Want a **public** cryptographic hash function → ideally, this would be a "*random function*" (mathematically defined e.g. as random oracle), but cannot implement in practice that way

Requirements

- ☐ H(x) is relatively easy to compute for any given x
- ☐ one-way or pre-image resistant
  - computationally infeasible to find x such that H(x) = h
- ☐ second pre-image resistant or weak collision resistant
  - computationally infeasible to find $y \neq x$ such that H(y) = H(x) (for a given x)
- ☐ collision resistant or strong collision resistance
  - computationally infeasible to find any pair (x, y) such that H(x) = H(y)

Cryptographic hash function



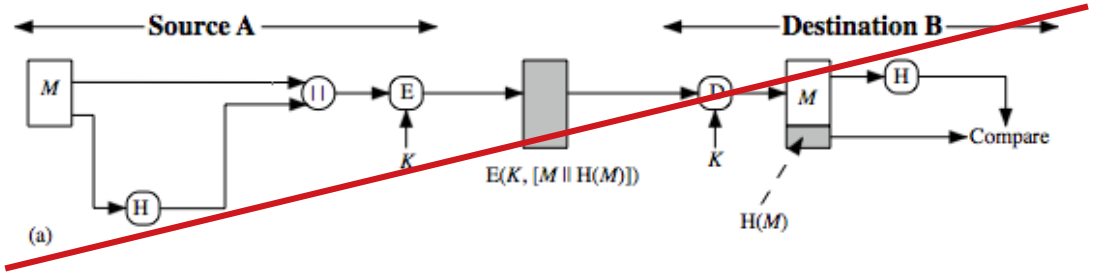
Security of hash functions

- There are two approaches to attacking a secure hash function:
  - cryptanalysis
    - exploit logical weaknesses in the algorithm
  - brute-force attack
    - strength of hash function depends solely on the length of the hash code produced by the algorithm

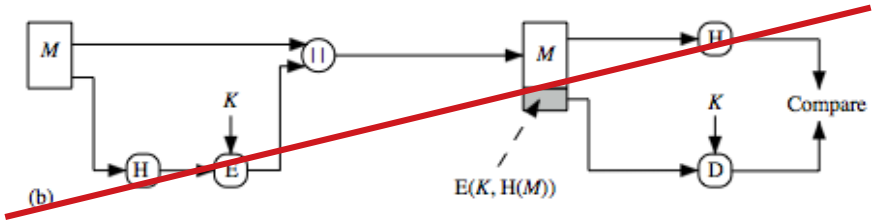
- SHA (v2/v3) most widely used hash algorithm

- Additional secure hash function applications:
  - passwords
    - **(slow + salted)** hash of a password is stored by an operating system
  - intrusion detection
    - store H(F) for each file on a system and secure the hash values
  - pseudorandom function (PRF) or pseudorandom number generator (PRNG)

Hash functions & Message authentication

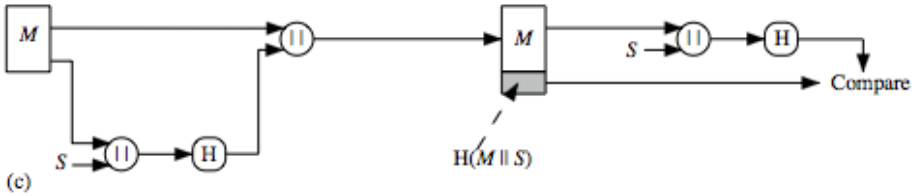


Message plus its hash are encrypted
→ Modifications must create two changes which also have to match, which is easy with stream ciphers

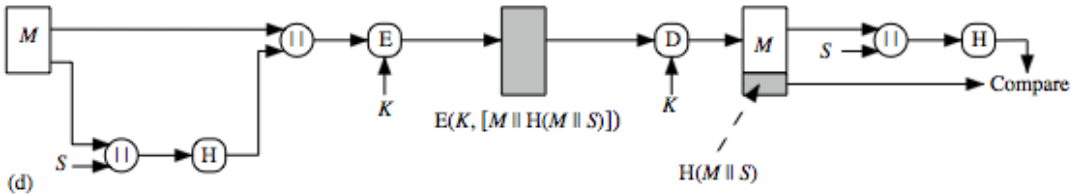


Cleartext message plus encrypted hash
→ "Signature" of message with symmetric/secret key, but need block cipher with appropriate block size

Non-repudiability cannot be guaranteed in any of these options!

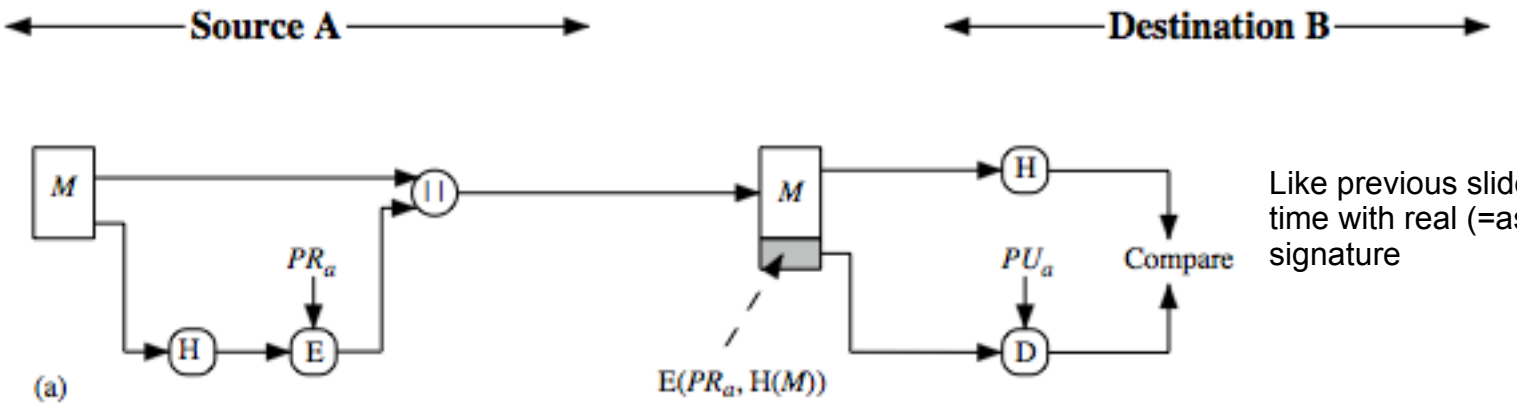


Message plus secret is hashed
→ "Signature" of message without symmetric or asymmetric cipher



Message plus its hash (including a secret) are encrypted
→ Encrypted message plus additional symmetric "signature"

Hash functions & digital signatures



Like previous slide, but this time with real (=asymmetric) signature

Provide non-repudiability

Full Signature + encryption (symmetric or asymmetric)

Secure Hash Algorithm (SHA-1)

- SHA originally designed by NIST & NSA in 1993
- Was revised in 1995 as SHA-1
- US standard for use with DSA signature scheme
  - standard is FIPS 180-1 1995, also Internet RFC3174
  - nb. the algorithm is SHA, the standard is SHS
- Based on design of MD4 with key differences: produces 160-bit hash values
- Since 2005 results on security of SHA-1 have raised concerns on its use in applications, based on 2015 results (on-the-way "freestart" collisions found) have to **consider it broken in terms collision-freeness**

(And don't even think about using MD4/5)

Revised SHA-2 standard

- NIST issued revision FIPS 180-2 in 2002
- Adds 3 additional versions of SHA
  - SHA-256, SHA-384, SHA-512
- Designed for compatibility with increased security provided by the AES cipher
- Structure and detail is similar to SHA-1 → hence analysis should be similar, but security levels are higher

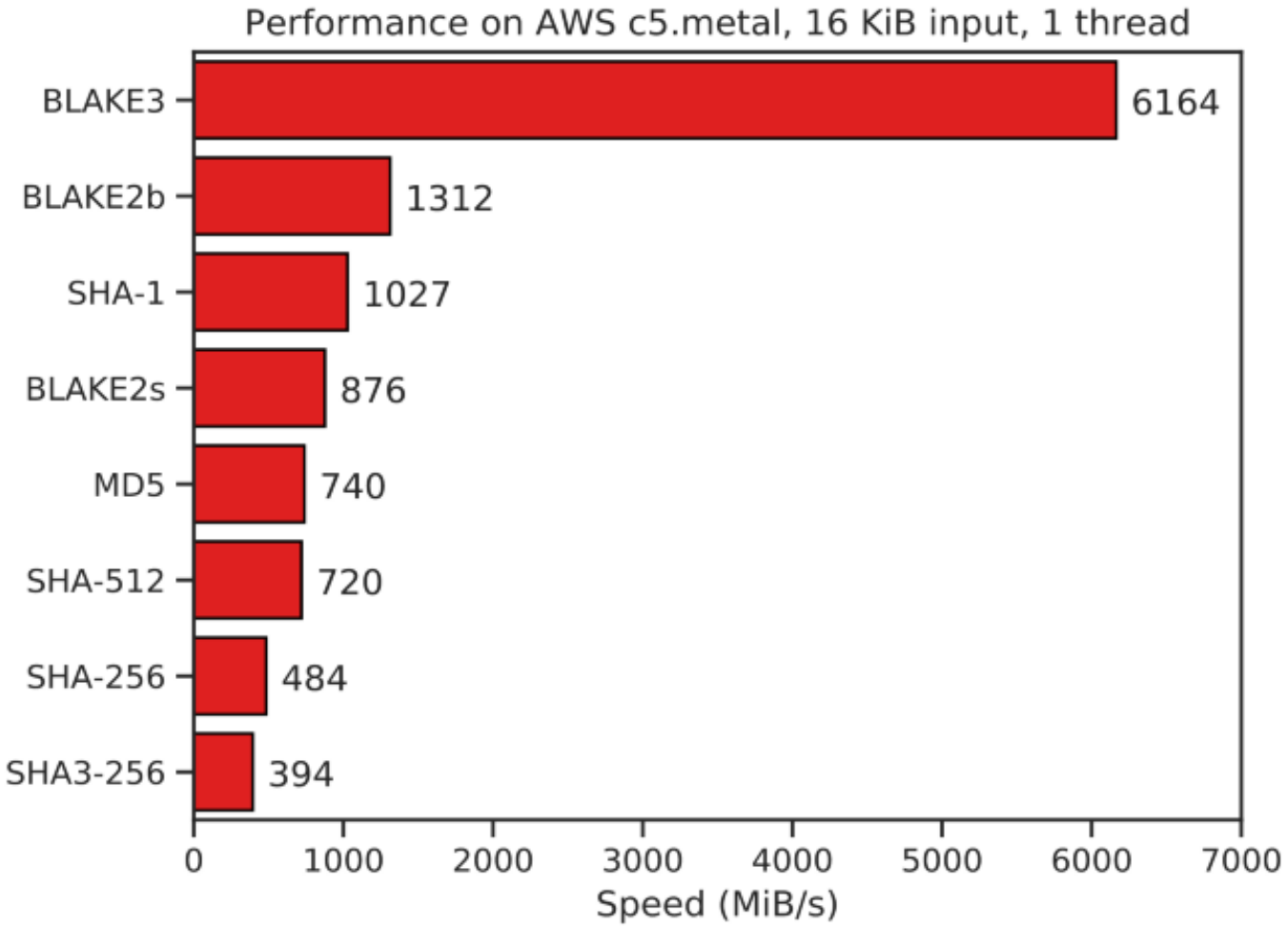
New SHA-3 standard

- SHA-1 needs to be considered broken **now**
  - https://sites.google.com/site/itstheshappening/ (paper at https://eprint.iacr.org/2015/967 from Oct. 2015)
  - 2017: Two PDF documents, both valid, same SHA-1, different content
- SHA-2 (esp. SHA-512) seems secure now, but may not remain
  - shares same structure and mathematical operations as predecessors
  - NIST competition for the SHA-3 next generation hash started in 2007
- SHA-3 process started to replace SHA-2: same hash sizes, online
- As of 2.10.2012, NIST announced that **Keccak** is now the SHA-3 standard after three rounds of selection
  - designed by team from Italy and (again, see Rijndael, ...) Belgium
  - different structure than SHA-2, therefore unlikely that cryptanalytic attacks will influence both SHA-2 and SHA-3 at the same time
  - details: http://keccak.noekeon.org/

More (presumably) secure hash functions exist

- BLAKE3
  - based on ChaCha stream cipher design
  - suggested in 2020 to improve on BLAKE2 (from 2012) and BLAKE (submitted to NIST competition in 2008 like Keccak)
  - compatible output sizes
    - BLAKE-256 uses 32-Bit words internally, produces 256 bits digest
    - BLAKE-512 uses 64-Bit words internally, produces 512 bits digest
    - truncated versions for producing 224 and 384 bits
  - assumed to have similar security level to SHA-3, but significantly faster
    - BLAKE3 internally uses a binary tree structure and thus parallelizes well
  - Argon2 uses BLAKE2b for password hashing
  - for details see https://github.com/BLAKE3-team/BLAKE3 and https://github.com/BLAKE3-team/BLAKE3-specs/blob/master/blake3.pdf
- Note: both SHA-3 (Keccak) and BLAKE2/3 are not susceptible to length extension attack

Performance comparison



[Figure taken verbatim from https://github.com/BLAKE3-team/BLAKE3]

Message Authentication Code (MAC)

- A MAC is a cryptographic checksum
  MAC = $C_K(M)$
  - condenses a variable-length message M using a secret key K to a fixed-sized authenticator
  - depending on both message and (secret) key
  - like encryption though need not be reversible
- Is a many-to-one function
  - potentially many messages have same MAC
  - but finding these needs to be very difficult
- Appended to message as a **signature** (but **both** sides know the key!)
- Receiver performs same computation on message and checks it matches the MAC
- Provides assurance that message is unaltered and comes from sender: **integrity** and **authenticity** → protects against active attacks
- Can use conventional cryptography with symmetric keys

Message authentication codes

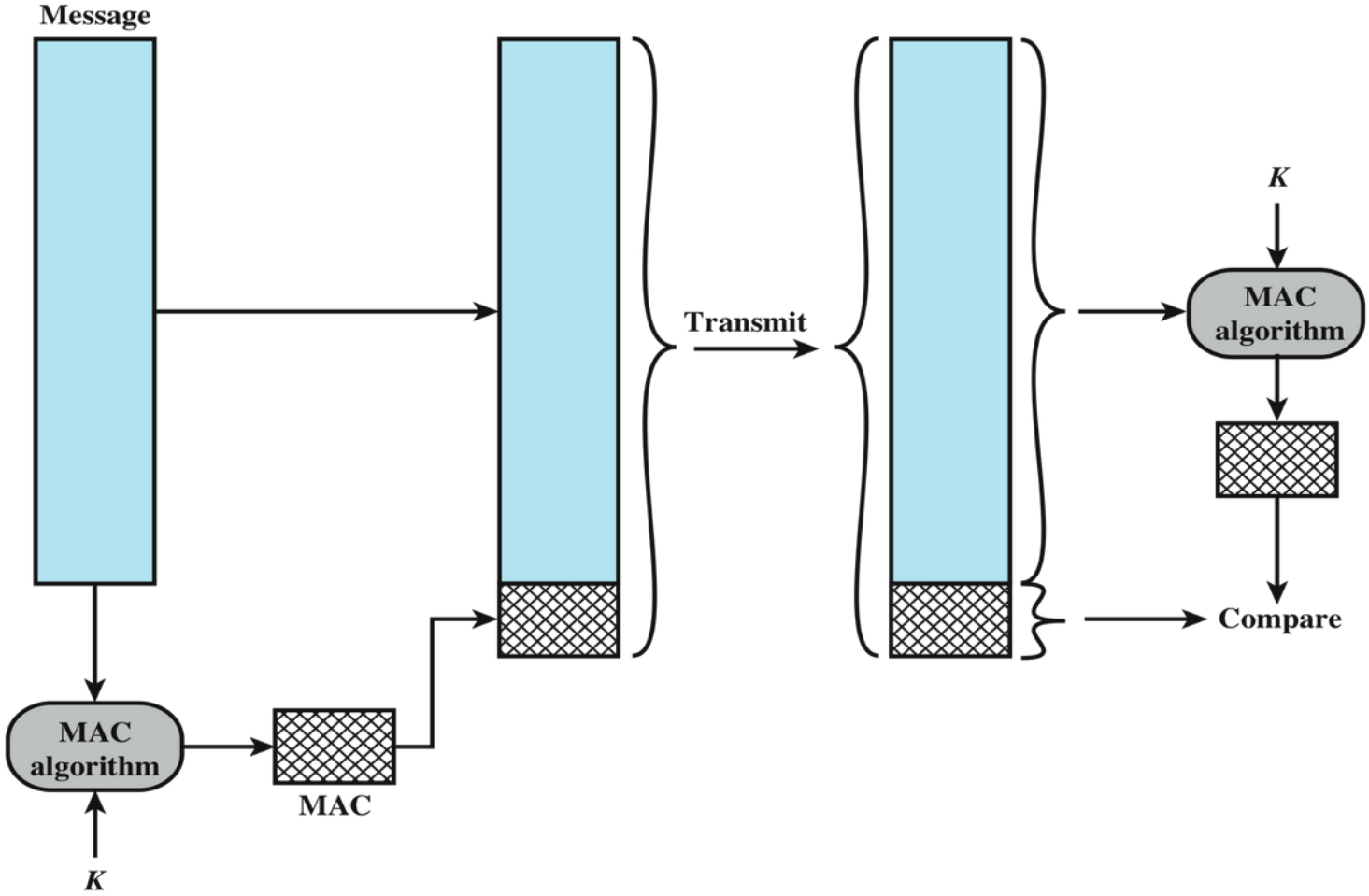


Figure 2.4 Message Authentication Using a Message Authentication Code (MAC). The MAC is a function of an input message and a secret key.

Message authentication codes

- As shown the MAC provides authentication
- Can also use encryption for secrecy
  - generally use separate keys for each
  - can compute MAC either before or after encryption
    - previously: ~~is generally regarded as better done before~~
    - currently: first encrypt, then MAC (because of padding attacks)
- Why use a MAC?
  - sometimes only authentication is needed
  - sometimes need authentication to persist longer than the encryption (e.g. archival use)
  - **Encryption does, in the general case, not provide implicit integrity protection (cf. stream cipher attack on cipher text)!**
- Note that a MAC is not a digital signature according to most common usage of the term, because it **does not offer non-repudiability**

Security of MACs

Like block ciphers have:

- **Brute-force** attacks exploiting
  - strong collision resistance hash have cost $2^{m/2}$
    - 128-bit hash is vulnerable, 160-bit better, but don't use less than 256-bit
  - MACs with known message-MAC pairs
    - can either attack keyspace (cf. key search) or MAC
    - at least 256-bit MAC is needed for standard security level (Birthday attacks)
- **Cryptanalytic attacks** exploit structure
  - like block ciphers want brute-force attacks to be the best alternative
  - more variety of MACs so harder to generalize about cryptanalysis
- Need the MAC to satisfy the following:
  - knowing a message and MAC, is infeasible to find another message with same MAC
  - MACs should be uniformly distributed
  - MAC should depend equally on all bits of the message

Keyed hash functions as MACs

- Want a MAC based on a hash function
  - because hash functions are generally faster
  - crypto hash function code is widely available
- Hash includes a key along with message
- Original proposal:

  `KeyedHash = Hash(Key|Message)`

  - some weaknesses were found with this, e.g. **message extension attack**
- Eventually led to development of HMAC

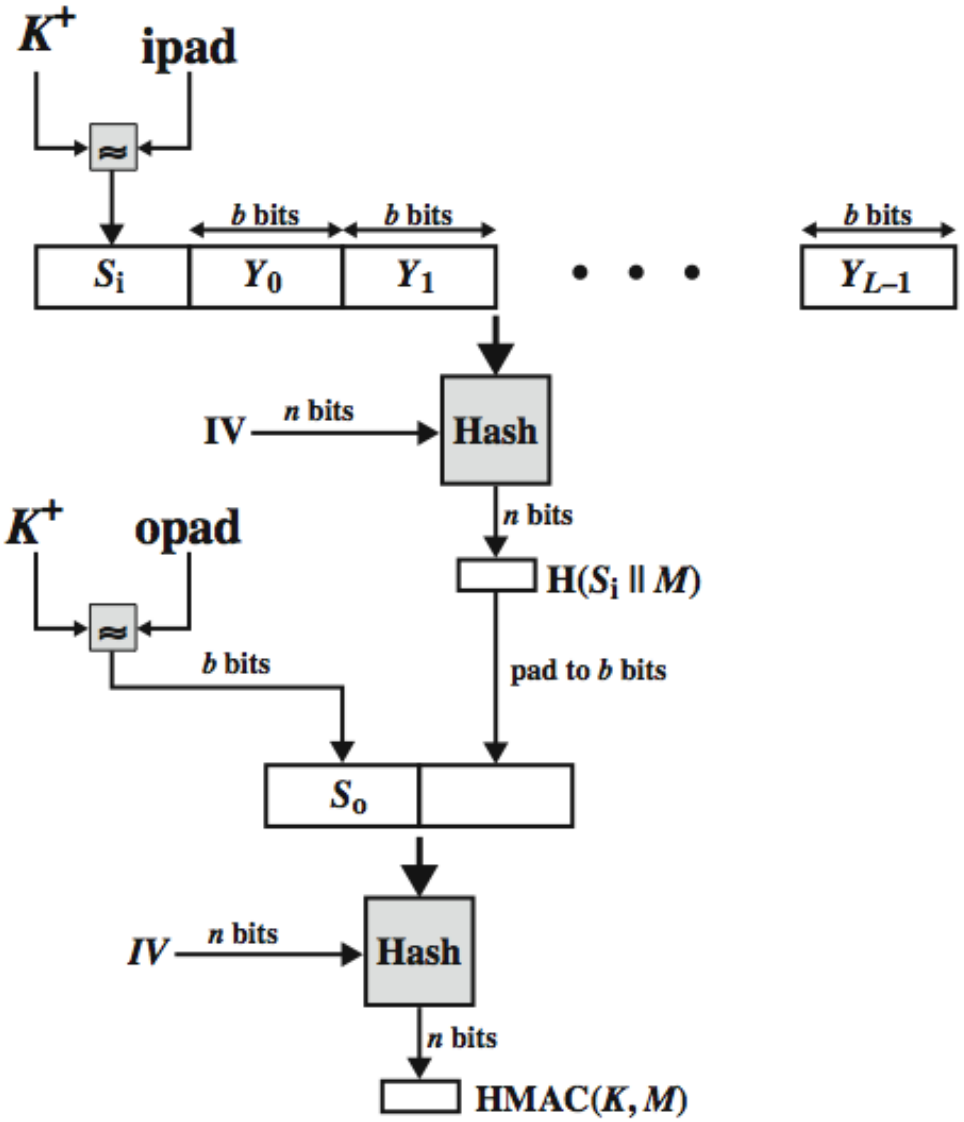
HMAC

- Specified as Internet standard RFC2104
- Uses hash function on the message:

$$HMAC_K(M) = Hash[(K^+ \text{ XOR } opad) \,||\, Hash[(K^+ \text{ XOR } ipad) \,||\, M)]\,]$$

  - K is key padded with 0's on right to block size of the hash function
  - opad/ipad: specified padding constants: 0x5C...5C / 0x36...36
- Overhead is just one more hash calculation than the message needs alone (= process three hash blocks more; two more than simple version from previous slide)
- Any hash function can be used
  - not: MD5, SHA-1, RIPEMD-160, Whirlpool,
  - use: **SHA-2**, **SHA-3**, **BLAKE2**, **BLAKE3**

HMAC overview

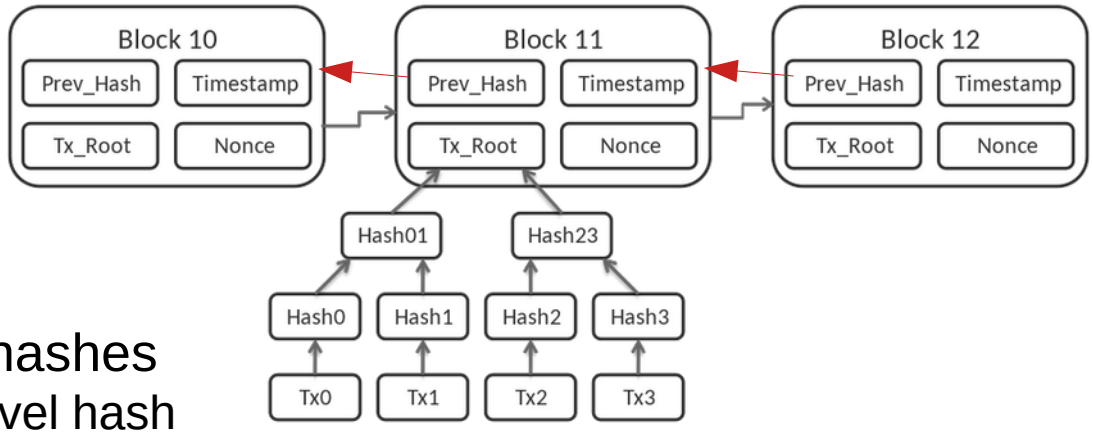


Authenticated encryption combinations

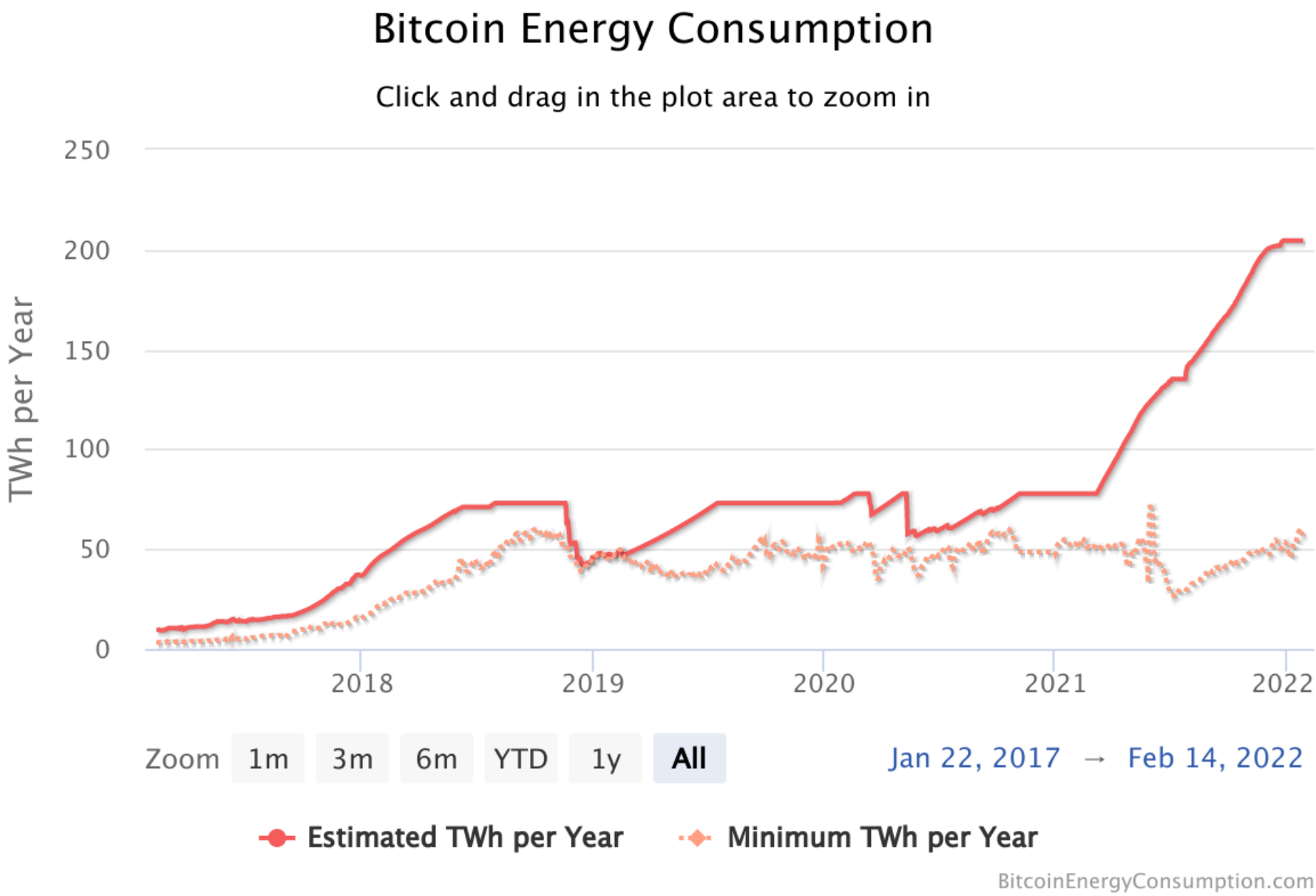
- Simultaneously protect confidentiality and authenticity of communications
  - often required but usually separate
- Approaches:
  - hash-then-encrypt: E(K, (M || H(M))
  - MAC-then-encrypt: E(K2, (M || MAC(K1, M))
    → Padding Oracle and Vaudenay attack (S. Vaudenay: “Security Flaws Induced by CBC Padding Applications to SSL, IPSEC, WTLS, …”)
    http://codeinsecurity.wordpress.com/2013/04/05/quick-crypto-lesson-why-mac-then-encrypt-is-bad/
    http://www.thoughtcrime.org/blog/the-cryptographic-doom-principle/
  - **encrypt-then-MAC: (C=E(K2, M), T=MAC(K1, C))**
  - encrypt-and-MAC: (C=E(K2, M), T=MAC(K1, M))
  - **best to use an AEAD mode (e.g. OCB, CCM, GCM) to combine encryption and MAC in one step and avoid this decision!**
- Decryption / verification straightforward

Source of figure: https://commons.wikimedia.org/wiki/File:Bitcoin_Block_Data.svg

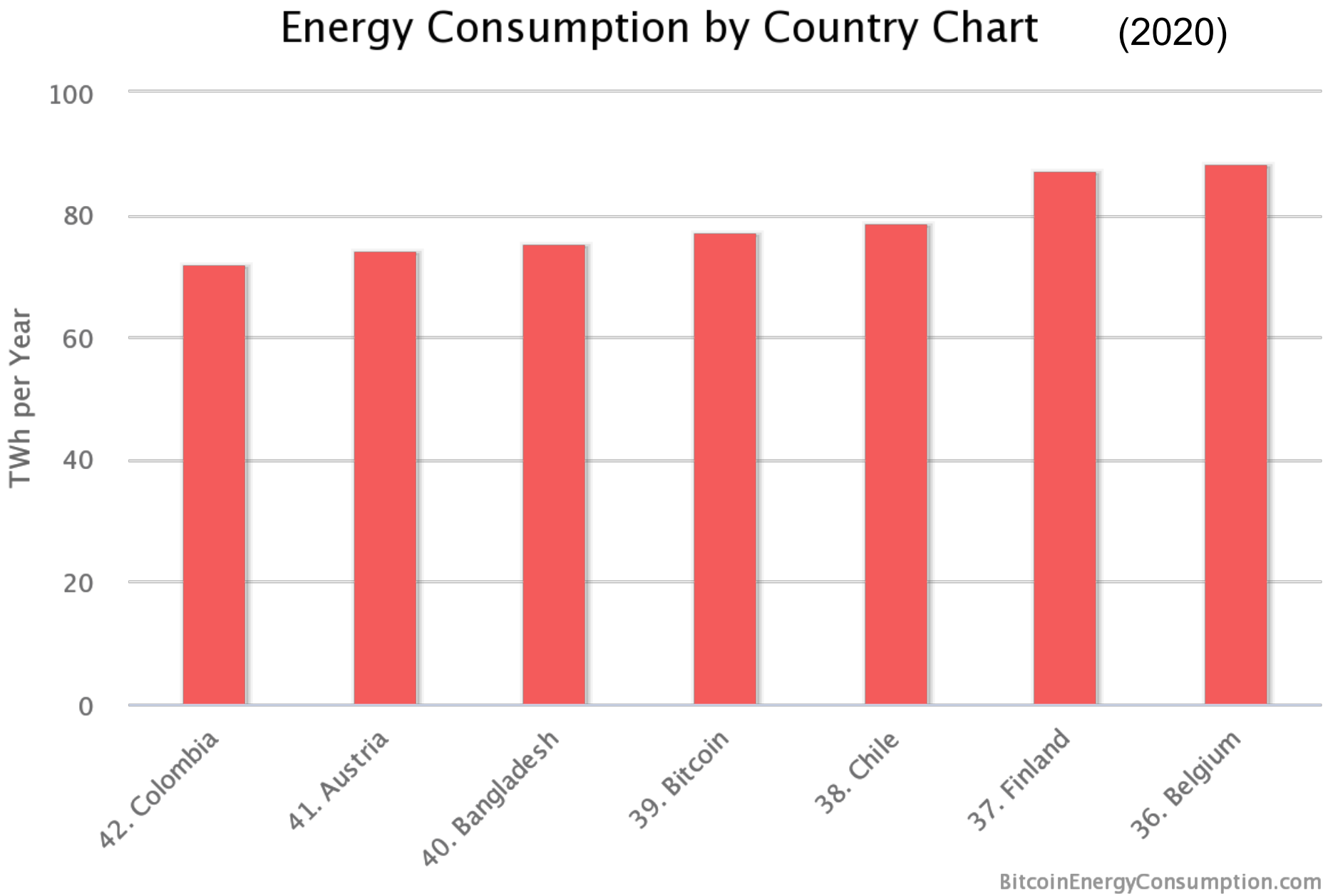
Blockchain



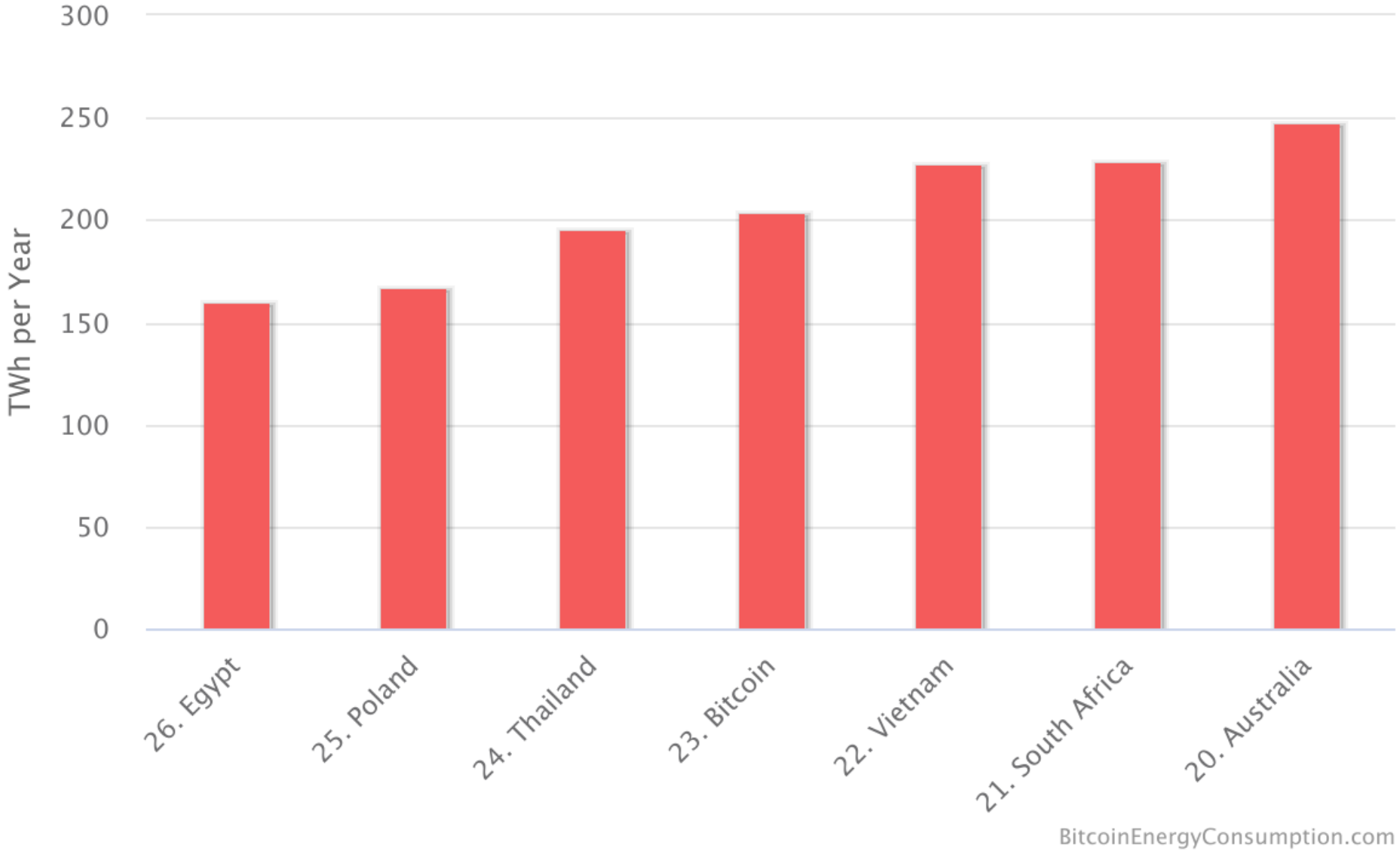
- Data structure based on hashes
  - next block includes top-level hash of previous block → chaining of blocks
  - each block contains (hashes to) data plus some meta-data (e.g. timestamp)
- If last block hash is trusted, can verify all preceding blocks
- Questions for practical use:
  - Where to store all blocks?
    - Bitcoin uses peer-to-peer network to distribute new blocks, every node stores whole chain
  - How to update last hash pointer, i.e. how to select newest block?
    - Bitcoin uses proof-of-work by having to brute-force hash challenges (cf. Nonce)
- Details:
  - https://cs251.stanford.edu/
  - https://github.com/matthewdgreen/blockchains/wiki/Course-Syllabus-2020



Source: https://digiconomist.net/bitcoin-energy-consumption


Energy Consumption by Country Chart
(2020)
100
80
60
40
20
0
TWh per Year
42. Colombia
41. Austria
40. Bangladesh
39. Bitcoin
38. Chile
37. Finland
36. Belgium
BitcoinEnergyConsumption.com

Energy Consumption by Country (2021)



Random numbers

- Keys for public-key algorithms
- Stream key for symmetric stream cipher
- Symmetric key for use as a temporary session key or in creating a digital envelope
- Handshaking to prevent replay attacks
- Randomizing encrypted/MACed messages to make traffic/message analysis harder

Random number requirements

Randomness

- **Uniform** distribution: frequency of occurrence of each of the numbers should be approximately the same
- **Independence**: no one value in the sequence can be inferred from the others

Unpredictability

- Each number is statistically independent of other numbers in the sequence
- Opponent should not be able to predict future elements of the sequence on the basis of earlier elements

Random versus pseudorandom

- Cryptographic applications typically make use of algorithmic techniques for random number generation
  - algorithms are deterministic and therefore produce sequences of numbers that are not statistically random
- Pseudorandom numbers are:
  - sequences produced that satisfy statistical randomness tests
  - likely to be predictable
- True Random Number Generator (TRNG):
  - uses a nondeterministic source to produce randomness
  - most operate by measuring unpredictable natural processes
    - e.g. radiation, gas discharge, leaky capacitors, resistor noise
  - increasingly provided on modern processors

Entropy

From Wikipedia articles:

- “In **thermodynamics**, **entropy** (usual symbol S) is a measure of the number of specific ways in which a thermodynamic system may be arranged, commonly understood as a measure of disorder.”
- “In **information theory**, (Shannon) **entropy** is the average amount of information contained in each message received. Here, message stands for an event, sample or character drawn from a distribution or data stream.”
- In **computing**, **entropy** is the randomness collected by an operating system or application for use in cryptography or other uses that require random data.”

In most cases, entropy means "disorder" or “uncertainty”

Key management

Require secure key management for symmetric cryptography

- Initial key exchange
  - transfer
  - verification
- Update
- Revoke

And all of these steps can be hard!

Why key management?

- Only provably secure encryption: one-time pad (OTP)
- But: key length = plain text length, and key is not re-usable
- Thus: impractical key management
- **Symmetric encryption is the first step towards solving the key management problem: to shorten the key which needs to be kept secret.**

Shortening the key

- Transferring the key over Internet connections to create secure connections
- ... over insecure channels

⇒ Chicken-and-egg problem

- Why not try to shorten the key itself by encrypting it with a shorter key?
- Because this would lower the entropy

⇒ **require different (out-of-band) mechanism for key management**

Key management methods

- Classical courier-suitcase-handcuffs scenario
  - maybe slightly expensive...
- Paper + (ground/snail) mail
  - PIN and TAN codes
- Telephone
  - slow, error prone, and insecure
  - compromise between usability and security
- Other out-of-band channels
  - cable, laser, infra red, ultra sound, etc.
  - quantum “cryptography” → please call it QKD (quantum key distribution)
- **Asymmetric cryptography**

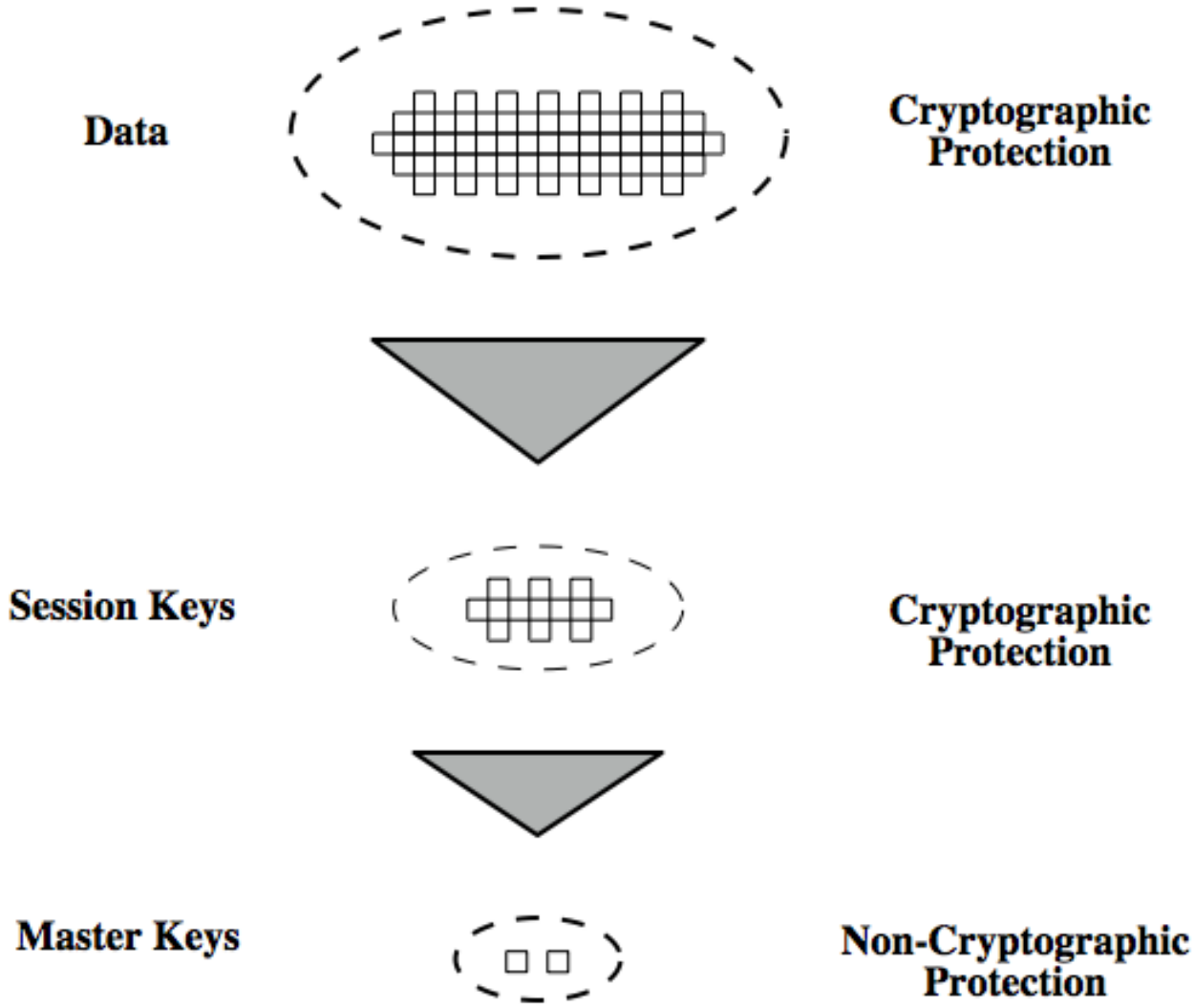
Hybrid cryptography system

- Combination of symmetric and asymmetric cryptography
  - symmetric: fast for bulk data encryption
  - asymmetric: (public) keys do not have to be kept and transmitted in secret
- **Session keys**
  - exchanged/established/managed by asymmetric cryptography
  - used as secret keys for symmetric cryptography
- Two ways to create session keys
  - establish using Diffie-Hellman key exchange
  - one party creates session key as random bit string, encrypted with public key of other party, optionally signed with private key of first party, and transmitted over insecure channels
- Session keys should not be re-used!
  - exception: “key continuation” methods (e.g. ZRTP)
  - but: better apply key continuation to symmetric “master” keys or to public keys

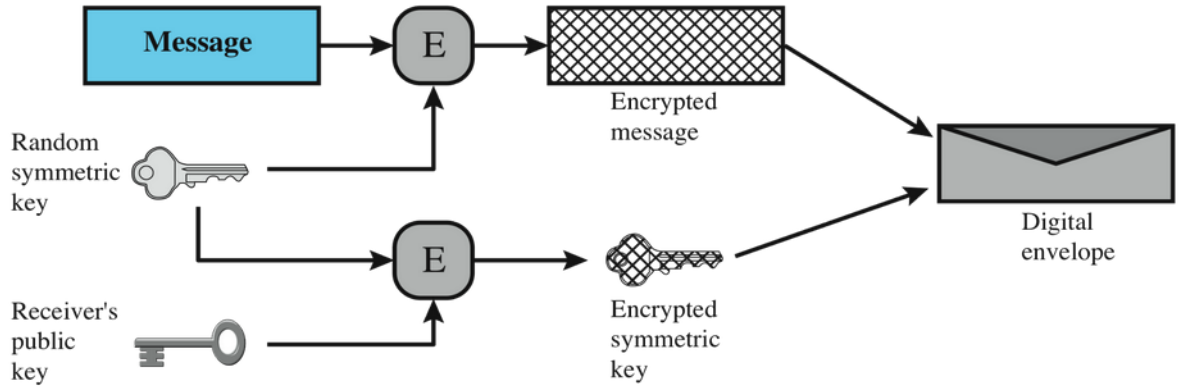
Key hierarchy

- Typically have a hierarchy of keys
- Session key
  - temporary key
  - used for encryption of data between users
  - for one logical session then discarded
- Master key
  - used to encrypt session keys
  - can be either asymmetric or symmetric (if other means for out-of-band transfer exist)

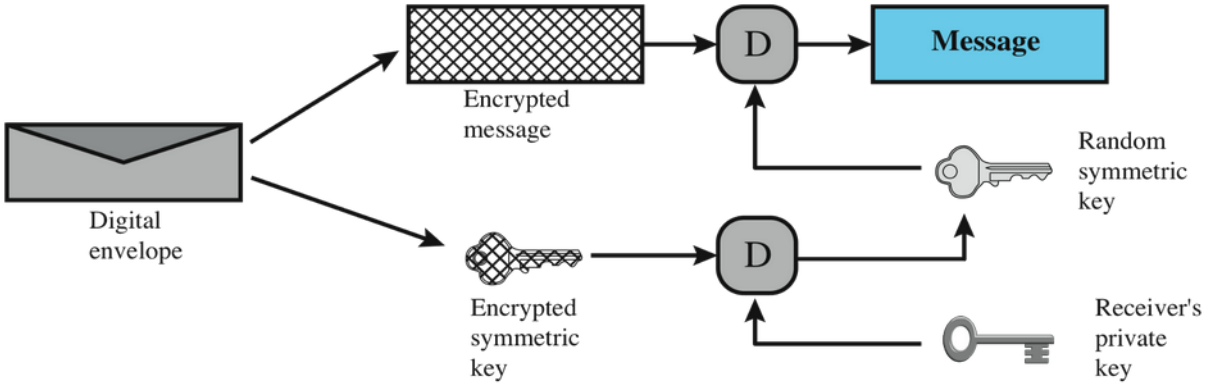
Key hierarchy



Hybrid system: digital envelope



(a) Creation of a digital envelope



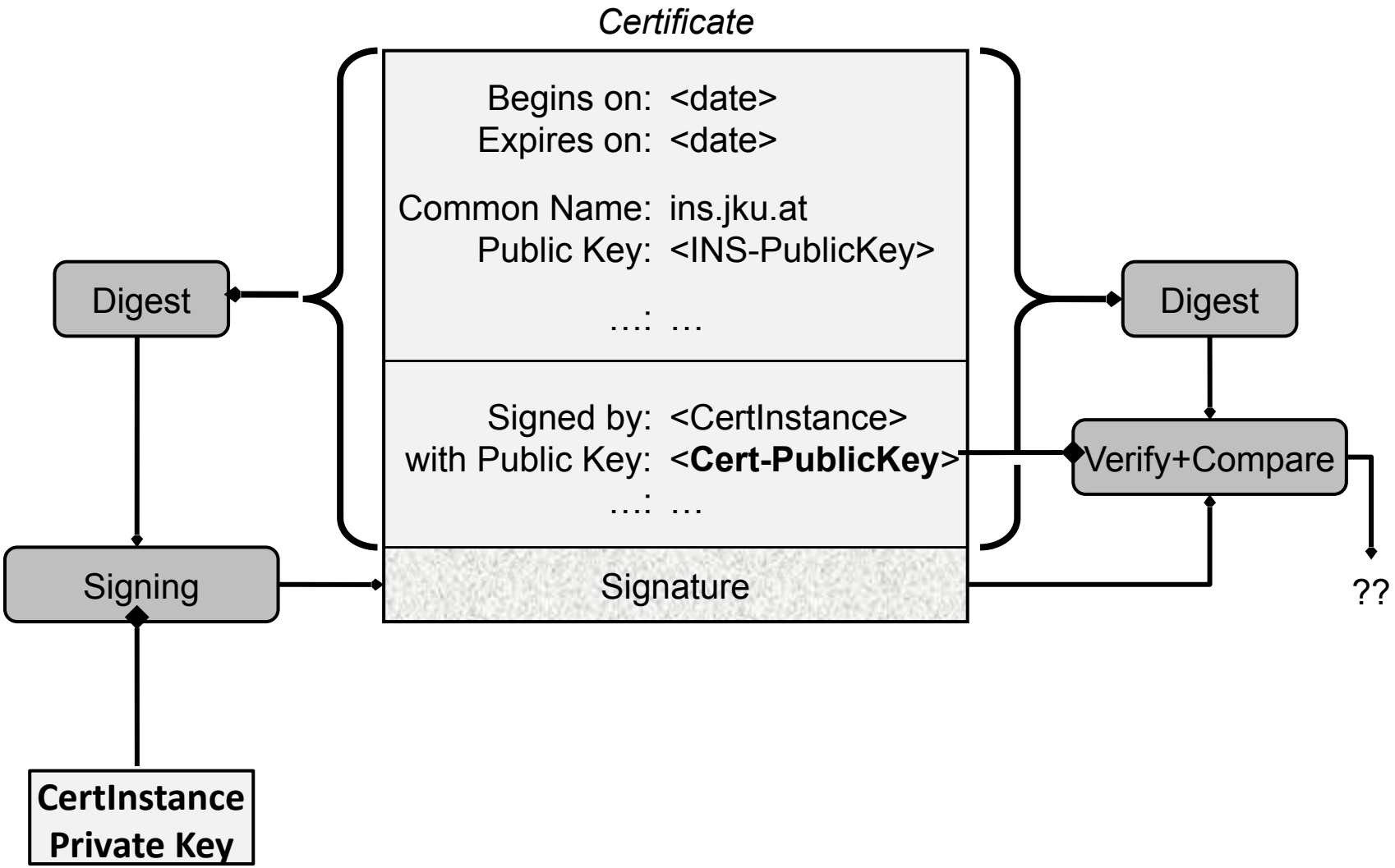
(b) Opening a digital envelope

Figure 2.9 Digital Envelopes

Public-key certificates

- Certificates allow key exchange without real-time access to public-key authority
- A certificate binds **identity** to **public key**
  - usually with other info such as period of validity, rights of use, etc.
- With all contents **signed** by a trusted Public-Key or **Certificate Authority (CA)**
- Can be verified by anyone who knows the public-key authorities public-key
- Examples: standard Public Key Infrastructure / CA companies
  - Verisign
  - Thawte
  - **Let's Encrypt**
  - ...

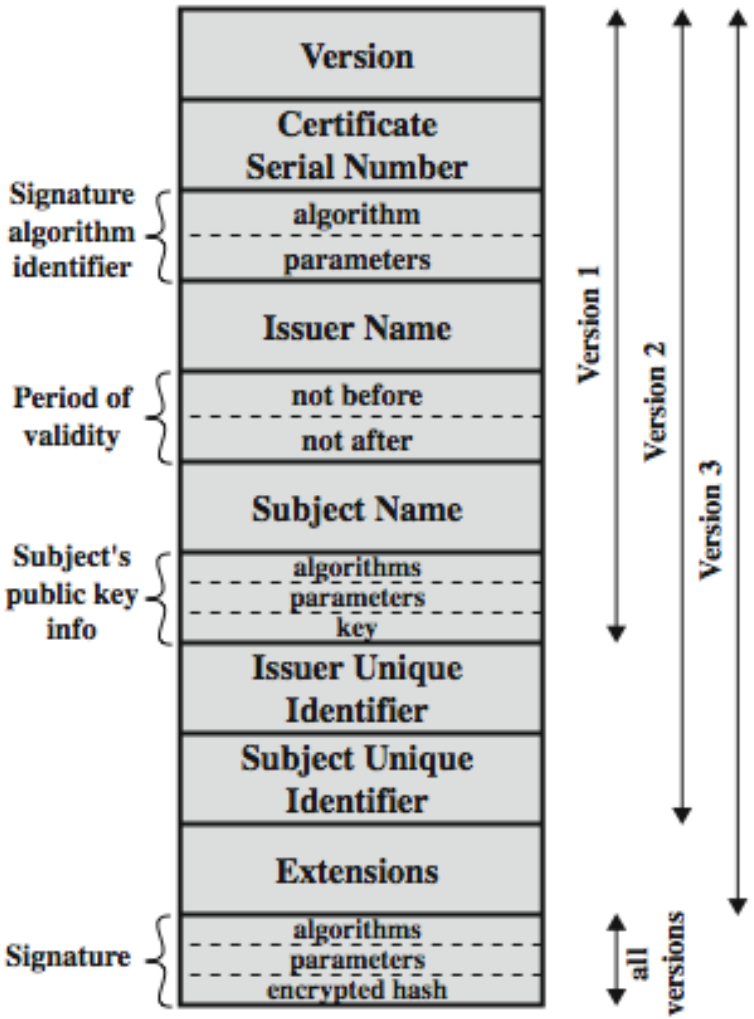
Public-key certificates



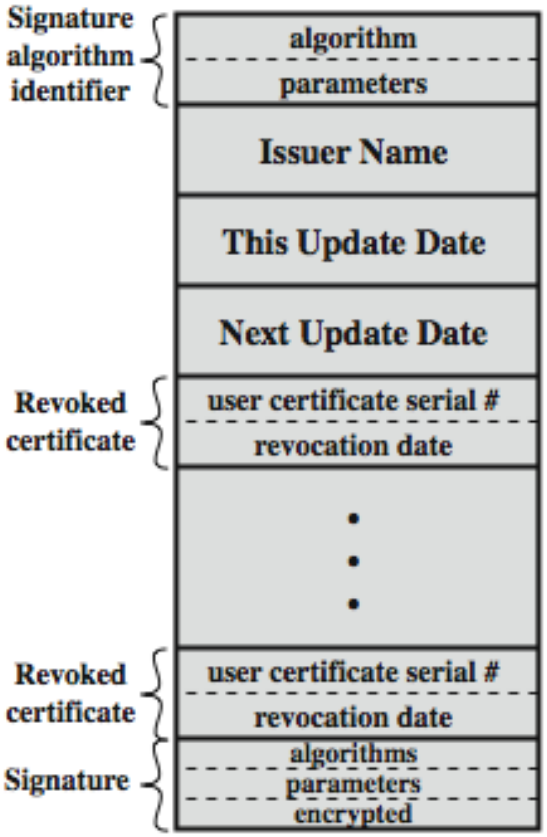
X.509 certificates

- Issued by a Certification Authority (CA), containing:
  - version V (1, 2, or 3)
  - serial number SN (unique within CA) identifying certificate
  - signature algorithm identifier AI
  - issuer X.500 name CA
  - period of validity TA (from - to dates)
  - subject X.500 name A (name of owner)
  - subject public-key info Ap (algorithm, parameters, key)
  - issuer unique identifier (v2+)
  - subject unique identifier (v2+)
  - extension fields (v3)
  - signature (of hash of all fields in certificate)
- Notation CA<<A>> denotes certificate for A signed by CA

X.509 certificates



(a) X.509 Certificate

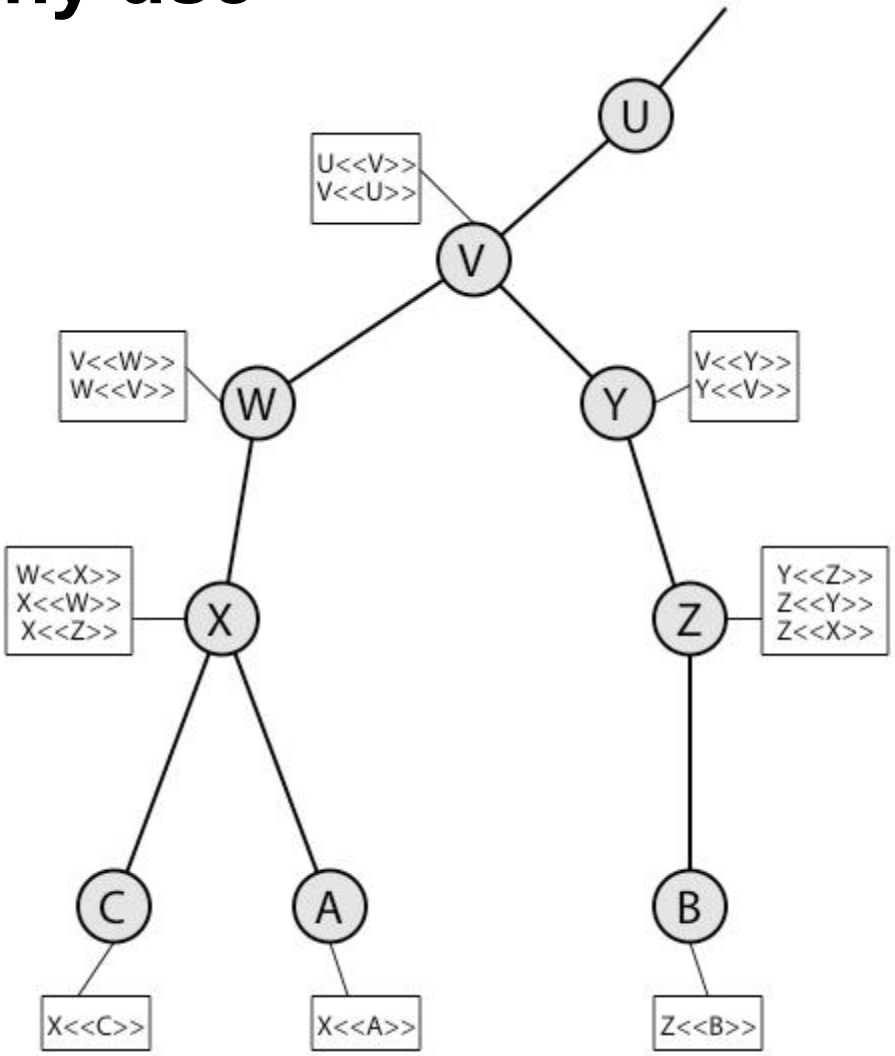


(b) Certificate Revocation List

CA hierarchy

- If both users share a common CA then they are assumed to know its public key
- Otherwise CAs must form a hierarchy
- Use certificates linking members of hierarchy to validate other CAs
  - each CA has certificates for clients (forward) and parent (backward)
- Each client trusts parents certificates
- Enable verification of any certificate from one CA by users of all other CAs in hierarchy

CA hierarchy use



Certificate revocation

- **Certificates have a period of validity**
- May need to revoke before expiry, e.g:
  - user's private key is compromised
  - user is no longer certified by this CA
  - CA's certificate is compromised
- CAs maintain list of revoked certificates
  - the **Certificate Revocation List (CRL)**
- Users should check certificates with CA’s CRL
- Still one of the biggest problems of PKIs

Problems with PKIs

- **All CAs can certify all hostnames/domains**
  - a single weak CA can break the whole PKI system
  - has happened in the past (see e.g. Comodo, DigiNotar, CNNIC, WoSign, TrustCor, ...)
- **All CAs are equally trusted** in the browsers (and other clients)
  - currently impossible to define which CAs are trusted by a client for Extended Validation (EV) and which are not
  - no mandatory standard to define which CAs are trusted for which domains/countries/etc. and which are not → RFC 6844 "*DNS Certification Authority Authorization (CAA) Resource Record*" from 2013 can be used optionally
  - but can remove a CA manually (=untrusted subtree)
- Many/most CAs only verify access to an email address for handing out certificates
- See e.g. http://lwn.net/SubscriberLink/663875/8e3238297b986190/

Partial solutions: Certificate pinning

- **Certificate pinning** allows to declare a binding between a server and a specific server certificate or a CA which is supposed to issue certificates for that server
  - can be implemented on the client (e.g. mobile app)
  - or server can instruct browser to pin with HKPK extension
    → also use HSTS to tell browsers to always use HTTPS instead of plain HTTP
  - tries to prevent misuse of malicious certificates for a server connection
- **Certificate transparency** tries to find different certificates being seen in the wild for the same server (also see various plugins for browsers for similar purpose) – orthogonal to pinning as a detection method

Partial solutions: DANE

- **DANE (DNS-based Authentication of Named Entities)** allows embedding X.509 certificates into DNS records
  - allows clients to query DNS for the certificates
  - if combined with DNSSec, can partially replace current PKI system (not for Extended Validation certificates)
  - can be combined with current PKI system by specifying CA allowed to issued certificates (certificate pinning in DNS)
  - See current RFC 6844 (https://tools.ietf.org/html/rfc6844)
- New CA effort: https://letsencrypt.org/
  - allows **automatic** (and free) provisioning of certificates to servers based on information from DNS and the web server itself
  - simple command-line tools to manage certificates directly on servers
  - automation is good → when it's done regularly, it is known to work!

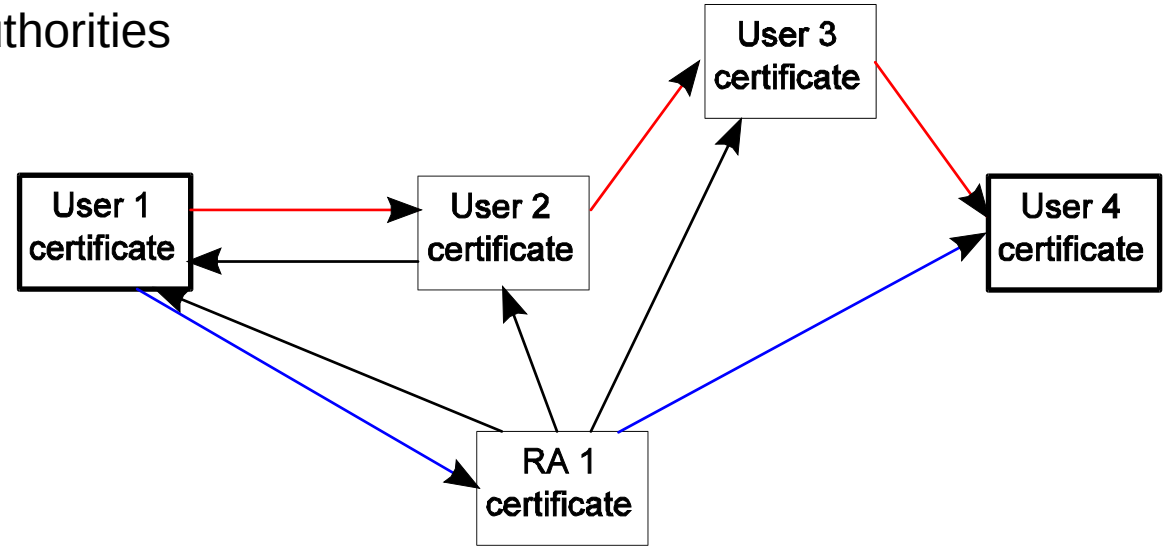
TLS server best operations practices

- Use certificates with secure hashes → SHA-256 or better
- Stay up-to-date with cipher suites (no RC4, no AES-CBC, no DH with <= 1024 Bits, …)
- If possible, keep private key on HSM (hardware security module)
- Patch/update HTTP server versions and crypto libraries whenever security updates are released
- … and many more

  Hint: check your servers (and browsers) with https://www.ssllabs.com/ssltest/ - many good tips to improve

Web of Trust (WoT)

- Alternative to PKI
  - no single root certificate
  - no distinction between user and CA certificates
  - users can “certify” other users
    → “I have verified that this public key belongs to the user with this name.”
  - special users may act as certification / registration authorities



Updating keys

Encryption and authentication keys need to be updated periodically

- When a maximum number of messages/bytes has been secured with the session key (statistical attacks, cryptanalysis)
- After a maximum lifetime (brute force attacks)
- After compromise

Possibilities

- Symmetric: just use a completely new key (re-keying)
  → all the previous applies
- Asymmetric: Need to re-transmit authentic public key (not likely)

Current best standard: *Signal* protocol, *Noise* as more generic version

Revoking keys

Asymmetric keys

- When a private key has been compromised (it is no longer private) or no longer in use
- Lifetimes of (self-) certificates
- Certificate revocation lists (CRLs)
- Online status checking (OCSP)

→ **One of the largest problems of PKIs, still practically unsolved**