**IY3660 Crypto**

Art of secret writing

Study of mathematical techniques related to aspects of information security such as confidentiality, data integrity, entity authentication, and data origin authentication

By **infrastructure** we mean the procedures, plans, policies, and management

Confidentiality – cannot be viewed by unauthorised entity

Integrity - assurance that data has not been altered (unauthorized/accidental)

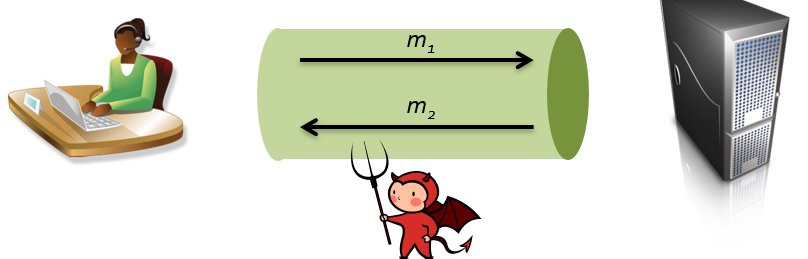
Entity authentication

Data origin authentication

Non-repudiation

\*Quantum Insert = active MITM drive-by attack on browser

Main App: Secure Comms



End to End (partial communication could be secure, but not all)

Security Goals:

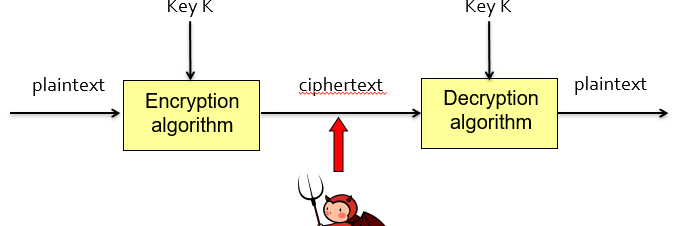
* Messages exchanged should remain confidential
* Hard to inject messages
  + Check origin
* Detect messages deletion
* Detect message reordering

Adversaries (active/passive)

* All data transferred can be seen
* Have control over network to delete, delay, modify, reorder network packets
* Can inject new network packets

MORE

* Ask specific messages m of his choice (chosen plaintext atk)
* Observe effects on network messages when inject packets of his own (chosen ciphertext atk) see error msg



Kerchoff’s principle

* Unbreakable
* Compromise of system = should not inconvenience correspondents
* Keys should be memorable/easy changed
* Transmissible by telegraph
* Portable/by single person
* Easy

Modern

* Security of cryptosystem should ONLY rely on secrecy of keys
* NOT on secrecy of system details

Obscurity = improve security

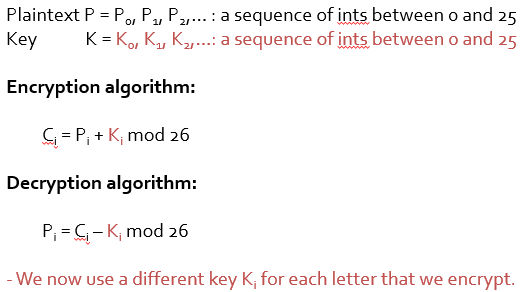
* Can + will be RE
* Still be secure even known
* Open specs = encourage review+analysis = improve confidence

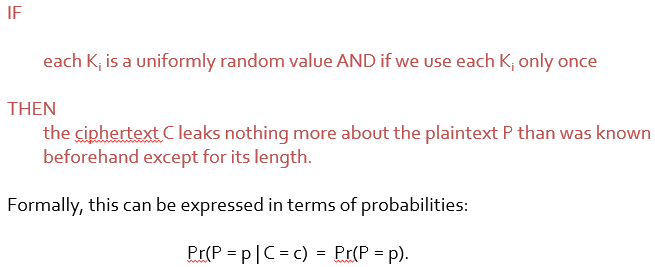
Caesar Cipher

= shifting letters left and right

* 25 possible key
* Freq Analysis

One Time Pad





Every possible 7-letter big cat can arise as a possible plaintext.

If the adversary knew *a priori* (in advance) that a big cat had been asked for, then the ciphertext tells him nothing new.

Unbreakable if:

Key K = needs as long as message

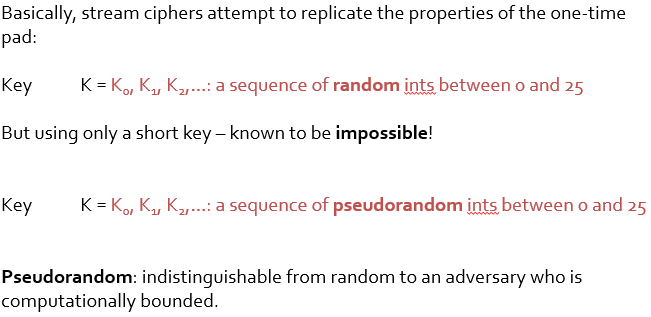
Key K = distributed in advance to communicating parties

Key = sent securely

Reused Key = fatal to security

* Subtract cipher text = leaving the diff of PT

**Stream Cipher**



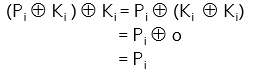
= takes as input KEY ( and possibly IV) and generates output as (Infinite) stream of bits (keystream)

* Bits, not int
* IV enable different key stream
* IV = fixed width + set to be counter in app

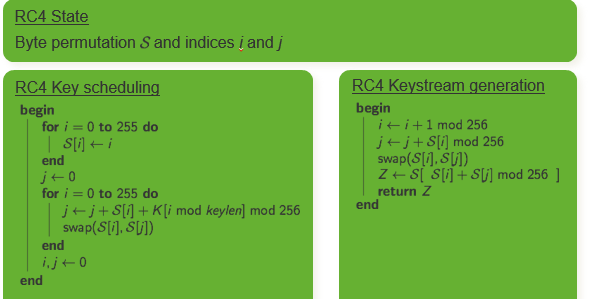
Encryption

* Decompose plaintext to bits
* XOR the plaintext and keystream

Decryption

* Decompose plaintext to bits
* XOR ciphertext with keystream
* 

RC4



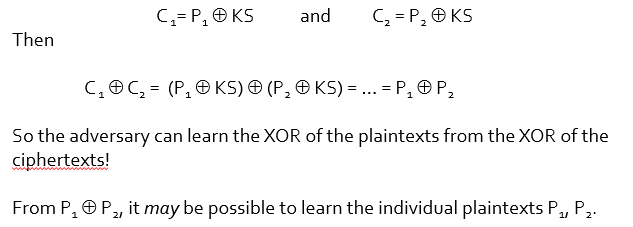
A5/1

* Used in hardware
* Sig amount of cryptanalysis

Modern ones:

* AES, 3DES
* eSTREAM
* ChaCha (variant of Salsa) = used in TLS (default in OpenSSH since 6.8)

**Bad Keystream REUSE**



* Statistics of plaintext, natural language

Security issues with stream ciphers

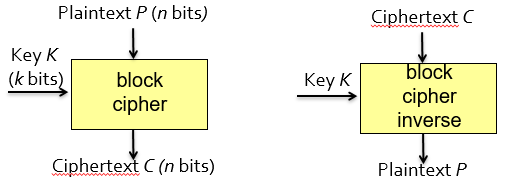
* WEP = use RC4 with 40bit key + 24bit IV
* Key = set once, left forever
* IV = usually counter
* Repeated keystreams

**Block Cipher**

= E / D blocks of bits: n bits at a time

DES = 64 bits, AES = 128 bits

E = control of key K of size *k* bits



E = encipher (not encrypting, can use for expansion of text)!

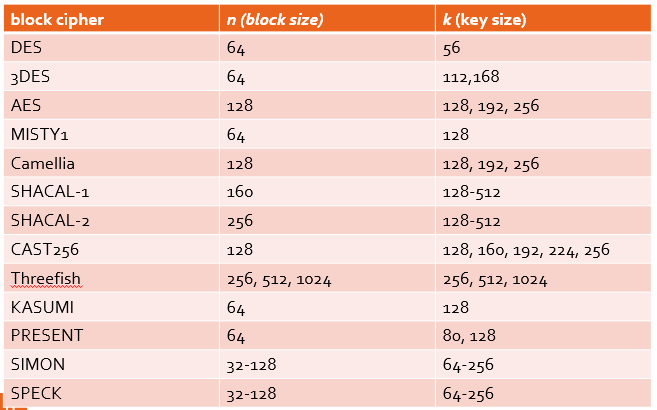
Block cipher with key length *k* and block size *n* consists of 2 ssets of efficiently computable bijections



such that Dk is the inverse of Ek for each K in {0,1}k

Usage:

* other block ciphers (e.g. E-D-E)
* encryption schemes (e.g. OCB, GCM)
* hash functions (e.g. Davis-Meyer mode)
* stream ciphers (e.g. CTR mode)
* message authentication codes (e.g. CMAC)
* pseudorandom bit generators (e.g. Fortuna).



**NIST** (US National Institute for Standards and Technology):

* AES, 3DES, Skipjack

**NESSIE** (New European Schemes for Signatures, Integrity and Encryption, EU project):

* AES, MISTY1, Camellia, SHACAL-2

**Cryptrec** (Japan):

* AES, 3DES, Camellia

**Factors while choosing a block cipher:**

* Key size (primary factor)
* Block size (secondary factor)
* Results of cryptanalysis
* Standardisation and support in crypto libraries
* Cost of implementation
  + Code size, number of transistors (GE), size of state
  + Potential for sharing code with other crypto primitives (in constrained environments)
* Runtime costs
  + Energy consumption, power consumption, throughput
  + Key agility (ease with which key can be changed/set up)
  + Hardware support (eg AES instructions on Intel and AMD processors)
* Implementation security
  + - Resistance to side-channel attacks (power, timing, …)

IF in doubt

* AES works well in most non-constrained applications, is widely supported in crypto-libraries, and has special hardware instructions on many modern processors.
* **Rule of thumb: if you need a block cipher, use AES.**

**Known plaintext attack:**

* adversary gets to **observe** many plaintext/ciphertext pairs (*P*,*C*) for a fixed key *K*, so: *C=EK*(*P*).

**Chosen plaintext attack:**

* adversary gets to **choose** many plaintexts *P* and is given the corresponding ciphertexts *C* under a fixed key *K*, so: *C=EK*(*P*).

**Chosen ciphertext attack:**

* adversary gets to **choose** many ciphertexts *C* and is given the corresponding plaintexts *P* under a fixed key *K*, so: *P=DK*(*C*).

**In each case, the target of the attack is the key *K.***

**Exhaustive Key Search =** if key size is small

**Realistic Attack**

IPSEC

* Used to encrypt IP datagrams and/or their payloads.
* Uses a block cipher as raw encryption mechanism.
* Adversary on network can see ciphertext blocks.
* Adversary knows many plaintext blocks, since many fields in IP packets, and the TCP and UDP packets they carry, are *fixed.*
* So the **known plaintext attack** setting makes sense.
* An adversary can inject into network traffic arbitrary IPsec packets containing ciphertext blocks of his choice.
* In some special cases, ICMP error messages may carry the decryptions of those blocks, and may be visible on the network.
* So the **chosen ciphertext attack** setting makes sense.

TLS

* Used to encrypt application layer data and transport it over TCP.
* Using remotely-delivered malicious Javascript running in victim’s browser, the adversary can arrange for arbitrary plaintext blocks *P* to be encrypted.
* The adversary can then see the ciphertext blocks on the network.
* So the **chosen plaintext attack** setting makes sense.

Exhaustive Key Search

* Ciphertext-only attack (IF plaintext = meaningful)
* embarassingly parallelisable
* may be other (statistical) attacks that are faster
* should have NO attack > exhaustive search

Advanced Attack

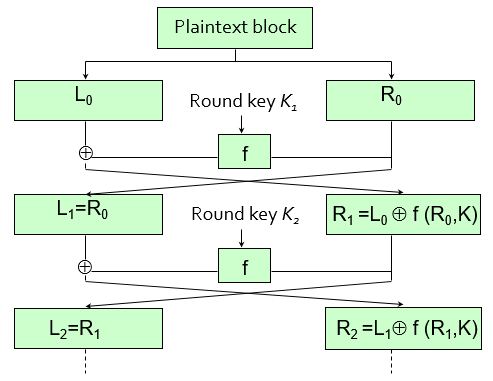
* Key recovery = best target
* IF not…
  + Related key attack
  + Requires Pseudorandom outputs

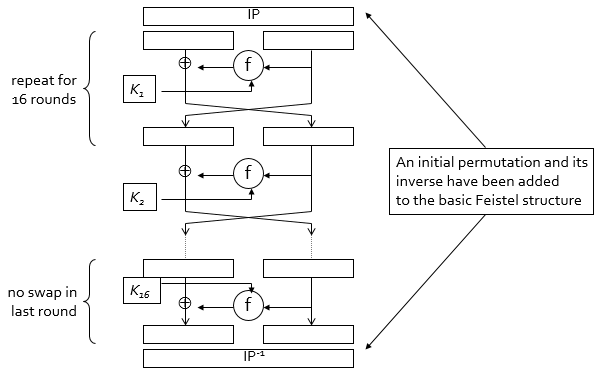
Construction methods

* Iteration of round function = universal
  + More rounds = stronger
  + Speed VS security
* Feistel Cipher (DES)
* SP Network
  + Each round func = built from substitution step followed by permutation (AES)

DES

* 3DES
* broken (but EKS = feasible)
* DES too small





**DES = bit manipulation**

* Slower in software than hardware

**Attacks + weakness FOUND**

* main issue = small key size
* not enough security
* EKS = average = 2^55 ops

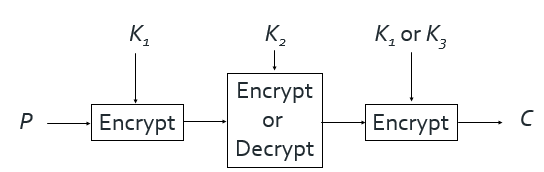
**DES Cracker**

* 92 × 109 keys/sec, 500k, 1993
* Now = 10k, 1 day

DES = well designed

* Best attack = exhaustive search

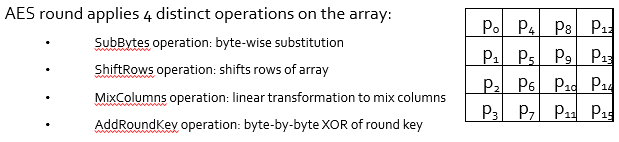
3DES



Double DES does not offer much additional security over single DES

AES

* Block size = 16 bytes (128 bits)
* Key size = 16 bytes (128 bits)
* 2^72 times harder than DES
* Substitution Permutation Network (SPN)



**Cryptanalysis**

* AES subject to intense analysis.
* Several attacks in related key setting or against reduced-round variants.
* First key recovery attack in 2011: workload of 2126.1 against 128-bit key version.

**Side-channel analysis**

* Berstein 2005: remote cache timing analysis against AES implementation in OpenSSL, requiring 200M chosen plaintexts and precise timing information.
* Osvik, Shamir, Tromer 2005: same-mahcine (co-resident) cache-timing analysis requiring 800 chosen plaintexts.

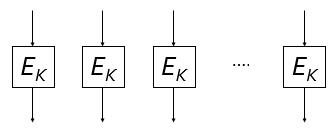
Modes of Operations

* Encrypt Flexible amount of data
* Different performance, Error propagation

ECB, CBC, CTR

Electronic Code Book (ECB)

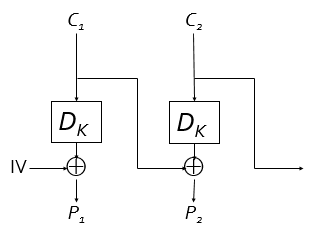
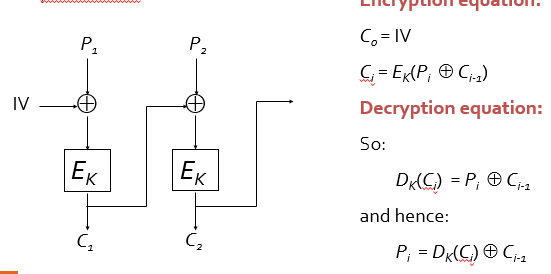
* Simplest



Leak quality of plaintext from ciphertext (info leakage)

* Parallelized encryption
* No error propagation

Cipher Block Chaining CBC



Error Propagation

IV = needed for decryption

* Must be random/unpredictable

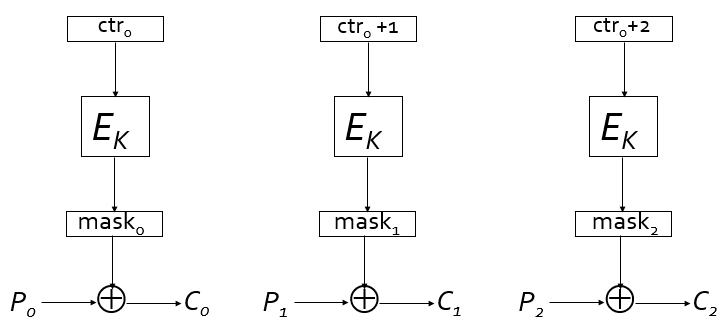
Padding = room for attack

N (if too small) = sweet32 attack

CBC = no integrity

CTR mode

* Uses a counter
* No padding required (last mark can be truncated)
* Parallelisable (can pre compute encr masks)

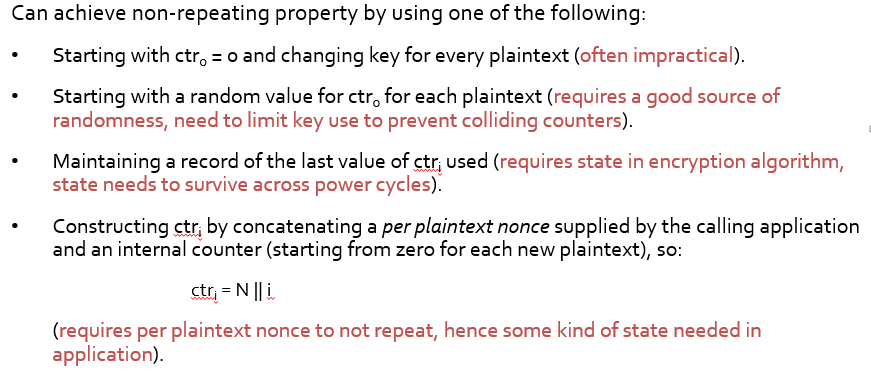


* STREAM CIPHER (turns block into stream)
* Block speed < stream speed (except special processors for AES)

No Err Propagation (bit flip in cipher= bit flip in plain)

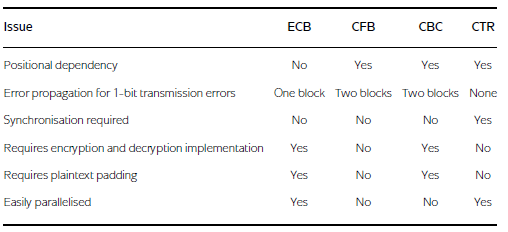
CTR = no integrity

Counter value = cannot repeat, cannot reuse



Security of CTR = pseudorandom of block cipher

* Output of random key & random string = indistinguishable

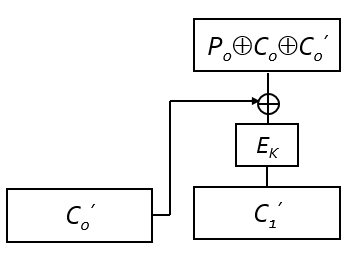


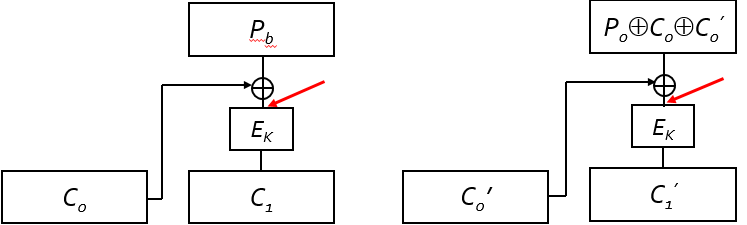
**Predicatable IV Attack**

adversary knows that either *P0* or *P1* has been encrypted

* Chosen plaintext attack

1. **predicts that *C0’***
2. Attacker now requests encryption of block   
   *P0* ⊕ *C0 (the IV)*⊕ *C0 ’(the prediction)*.
3. Attacker receives as ciphertext *C0’*, *C1’*.





* Hence, if *Pb = P0*, then *C1* =*C1’.*
* Otherwise, if*Pb = P1*, then *C1*≠ *C1’*.

Easy to multiple-block message

Can used in IV chaining (IV =last ciphertext block from prev message, CBC mode uses IV chaining)

Plaintext recovery (if low entropy, like password)

BEAST attack = full plaintext recovery

**Padding in CBC**

Pad to the n-bit size block 128 bits = 16 bytes

Use pad()

* pad() takes inputs in {0,1}\*
* pad() has outputs in {{o,1}n}\*
* pad() is necessarily expanding.
* pad() and its inverse both need to be efficiently computable.
* pad() may be randomised or deterministic.
* pad() adds (*t*+1) copies of byte value *t*, where 0 ≤ *t* < *n*/8, to bring message length up to multiple of *n* bits (*n*/8 bytes).

So, for AES, where *n*=128, we have *n*/8 = 16 and the possible pads added to messages are:

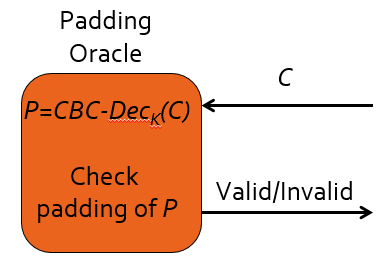
0x00

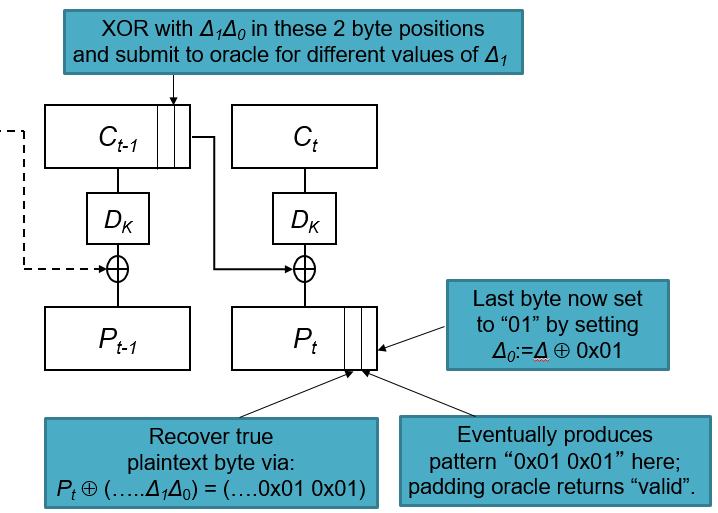
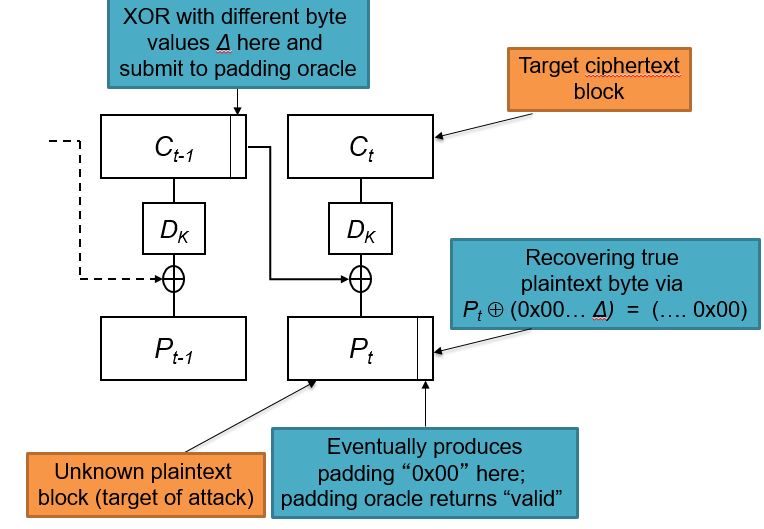
0x01 0x01  
 oxof oxof … 0x0f (16 copies of oxof)

After CBC mode decryption, message handling code needs to parse plaintext as: message||padding, then remove padding, and finally pass message to application.

* Should throw error
  + But crypto should not return error
* Adversary might be able to detect (logs, timing)

**Padding oracle error**

* Exploit padding error = full plaintext recovery



Avg of 128 (worst 256) for every byte (cos 2^8)

Found and fixed in 2002, lucky13 found again

Padding = heard, CBC with padding = generally vulnerable

Fixed by adding integrity with MACs

**HASH functions**

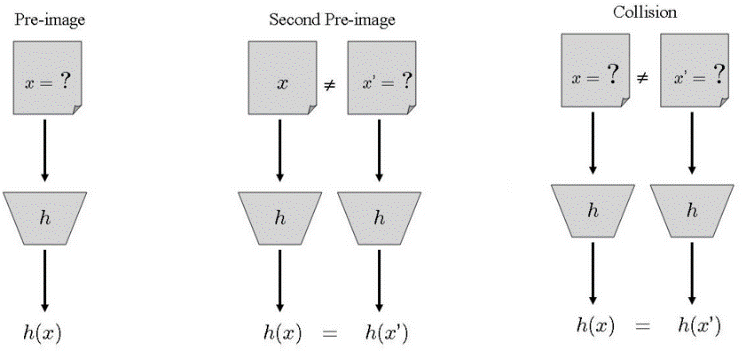
*n-*bit(cryptographic) hash function

H: {0,1}\* → {0, 1}n

* fixed-length hash value (sometimes called a message digest)
* behaving like random functions: given any input, output is a random *n-*bit string.

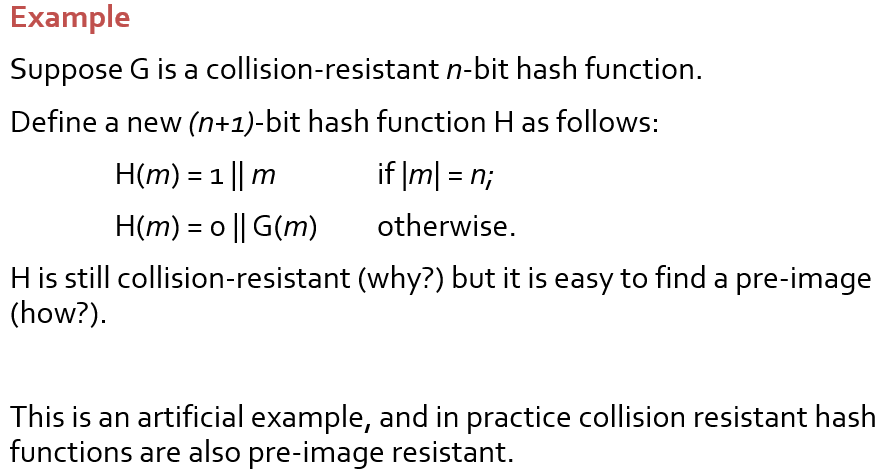
**Security Properties**

* Pre-image resistance (one way)
* 2nd image resistance
* Collision resistance

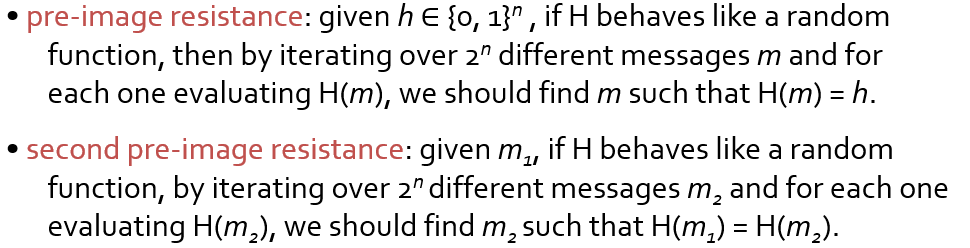
****

Collision resistance (implies) 2nd pre-image resistance

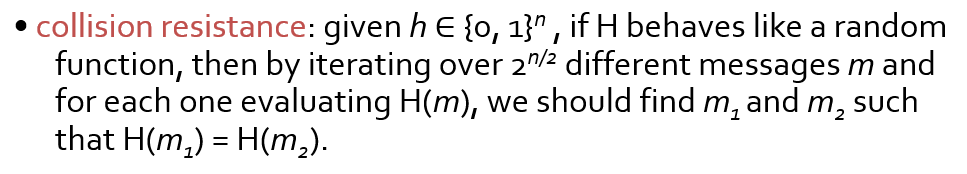
Collision resistance (does not imply) pre-image resistance



**Generic Attacks**

****

**Goal of designer =** no better attacks than generic atks



Birthday paradox = if elements are drawn at random from a set of size *s*, then after √*s*trials, we expect a collision in the sampled elements with about 50% probability

**HASH FUNCTIONS** (not all for cryptographic purpose)

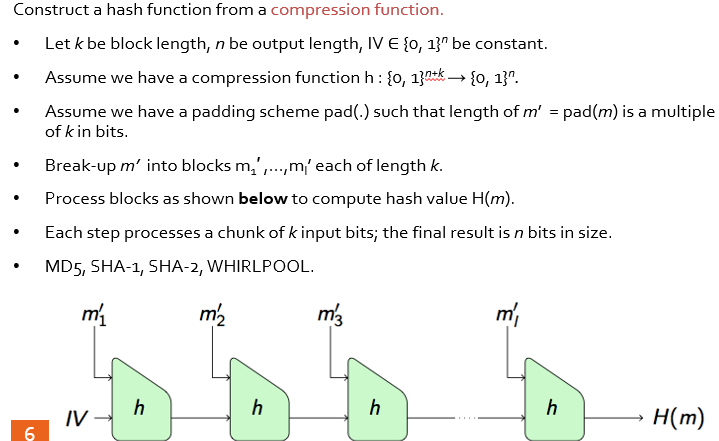
**\***insecure hashing in WEP (CRC hash used crypto hash)

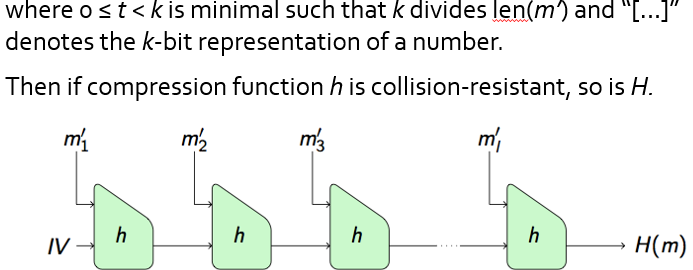
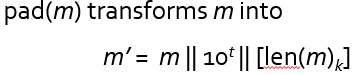
Design based on:

* Based on block ciphers
* Using modular arithmetic (mod n, not used)
* Dedicated design

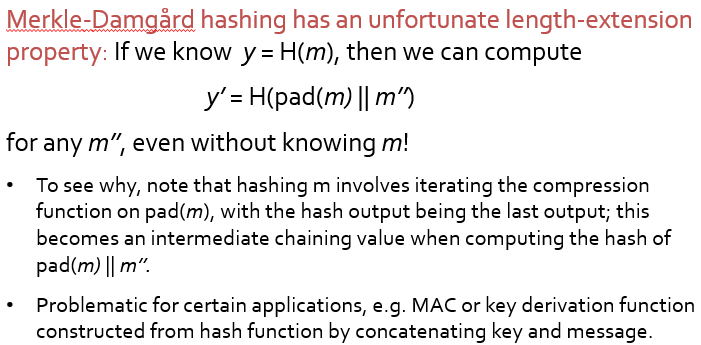
**Difficulty =** process **arbitrary length** messages

**Merkle-Damgard iterated hashing**



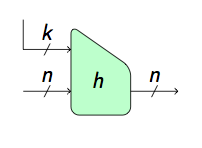
**Theorem (Security of the Merkle-Damgård construction)**

**Weakness = can add more iterations**

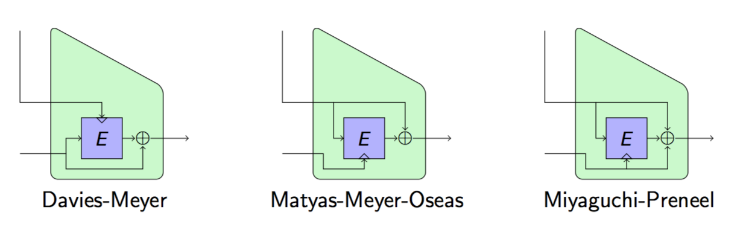


**Reduced our problem** of constructing a good hash function to that of constructing a good compression function *h*

* Interface requirement: h:{0,1}*n+k* → {0, 1}*n*, that is h maps *n+k* bits to *n bits.*
* Security requirements: collision resistant, one-way,…

**Construction from block cipher:**

* Davies-Meyer
* Some need additional key processing to map input to key space.
* Main issue is that hash output size = block cipher block size, so need specialised block cipher to avoid 2*n*/2 birthday attack.
* Secondary issue is that, in some constructions, message blocks are used to set key, so rekeying of block cipher needs to be fast.



Cant use AES , cos 128 bit then only 2^64 (birthday)

(**MD4**) **MD5**

128 bit hash  
RFC1321, used for file integrity checking

**(SHA-0) SHA-1**

160-bit hash  
Used in TLS/SSL, PGP, SSH, S/MIME, IPSec

**SHA-2** (**SHA224**, **SHA256**, **SHA384**, **SHA512**)

Recommended for US government use.

(**RIPEMD**) **RIPEMD-160**, **SHA-3**

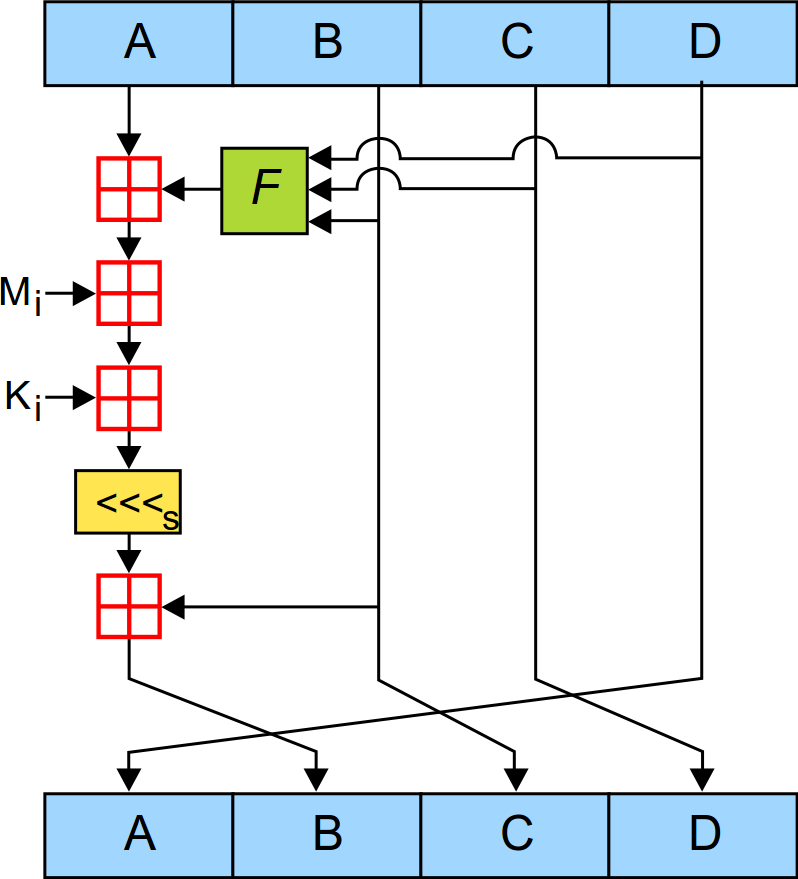
Standardisation recently completed by NIST, complementary to SHA-2 family (not a replacement).

**Whirlpool**

512-bit hash (based on modified AES).  
Used in TrueCrypt (open source encryption toolkit).

**MDx-family (Ron Rivest)**

* Merkle-Damgård hash with output size *n* equal to 128.
* Compression function uses (implicitly) a special block cipher in Davis-Meyer mode, *k* =512.
* Recall DM: message blocks → key; chaining variable → message.
* Block cipher has a round function operating on 4 x 32-bit words.
* Round function iterated 64 times.
* Use mod 232 additions, bit shifts, other simple bit operations (F)

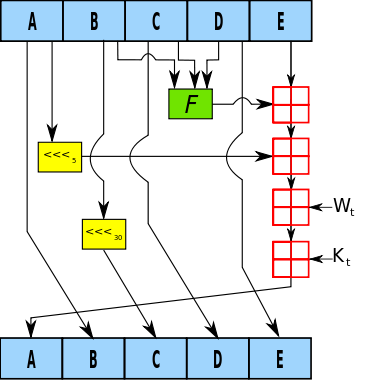


**Insecurity of MD5**

1991: MD4  
1993-1996: initial flaw  
2004: researcher proved  
2005: genereated X509 certs  
2008: obtained fake MD5 certs from verisign  
2012: malware uses fake cert

**Why so long to phase out**- once deployed, hard to phase out  
- backward compatibility  
- Require practical demo of attack  
- lack of understanding outside crypto  
- attacks only on collision, not break security properties

**SHA-1**



* Merkle-Damgård hash with output size *n* equal to 160 (20 byte output).
* Should provide 280 security against collision attacks.
* Compression function uses (implicitly) a special block cipher in Davis-Meyer mode, *k* = 512.
* Recall DM: message blocks → key; chaining variable → message.
* Block cipher has a round function operating on 5 x 32-bit words.
* Round function iterated **80** times.
* Use mod 232 additions, bit shifts, other simple bit operations (F).

**Insecurity of SHA-1**

1995: SHA-0  
2005: First attack on SHA-1 2^69  
2005-2015:reduce to 2^61  
2012: NIST retire SHA-1  
2014: Google penalise SHA-1 certs  
Oct 2015: first freestart collision attack on SHA-1

Freestart collision (weaker value)  
(cannot use any IV, fixed IV)  
Attacker gets to choose 2 different IVs   
(reality = fixed single value)

Estimated cost of full collision (75k-120k USD)

**SHA-2 (256)**

N= 256, k = 512

64 rounds (8x32 bit words using simple operations)

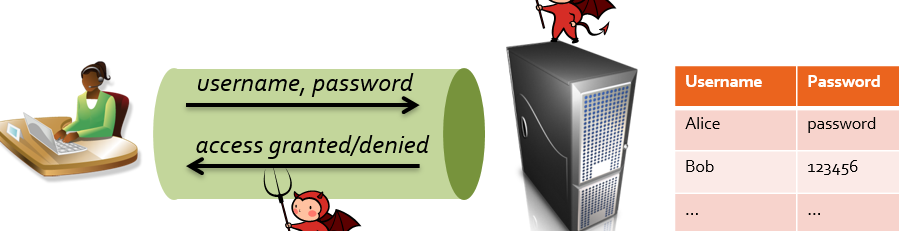
Slower than SHA-1

No attack faster than bday attack

**SHA-3 “Ketchak”**

New design approach  
‘giant bit permutation’ as key element in construction  
variable length output, throughput  
effiency/security tradeoffs  
Chosen for elegance, security margin, performance, flexibility (complement, not replace SHA2)

**Password hashing**



**Threat**: attacker breaks into the server and steal this database and then go on to impersonate users

**Solution**: store only password hashes in database.

* For each user, store H(password) instead of password.
* Server now hashes received password and compares to table.
* Intuition: attacker has to reverse hash to recover the password.

**Password breach**

* Absolute frequency: fraction amongst all accounts.
* Relative frequency: fraction amongst passwords of the same length.

**Password Security**collision = not important, just one-wayness  
sophistication = tuning, number-letter substitutions  
Use of GPUS, services  
Salting: store (salt, H(salt || password)) instead of H(password)

**Salt**

Add a random, account-specific value  
each account has to be attacked individually  
64 bits of salt is typical: enough to prevent collisions in random choices of salt values  
devalue pre-computation by an attacker.

Iterative hashing : PBKDF, bcrypt, scrypt, Argon2   
Slow down password cracking via dictionaries by iterating the hashing  
trade-off between security and performance of authentication  
scrypt(salt || password)

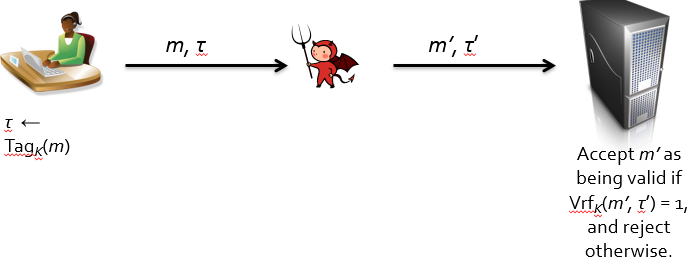
Encryption  
key management   
keys needs to be always available and heavily protected  
Careful selection of encryption method  
Enc*K,IV*(password)

MAC algos (for integrity)  
Dig Sig schemes

Key Derivation  
- derive one key from another key

MAC  
integrity and data origin authentication  
- no attacker should be able to forge messages

*Quantum Insert (hijack browser)*



share a **common secret key K**

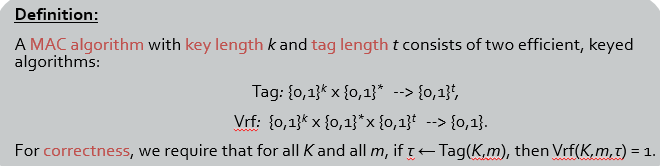
computes an **authentication tag τ** ← TagK(m) for each message m

* “MAC” or “MAC value” instead of “tag”.

Alice now sends **(m,τ)** to Bob, instead of just m.

Bob verifies the **integrity** and **data origin authenticity** of the received message/tag pair (m’,τ’) by using a corresponding Vrf algorithm.

**Security**: without knowing K, it should be **hard** for any adversary to create a pair (m’,τ’) such that VrfK(m’,τ’) = 1.



Correctness is a basic functionality requirement and **not** a security requirement.

**MAC unforgeability (informal):**

Cannot forge without knowing K

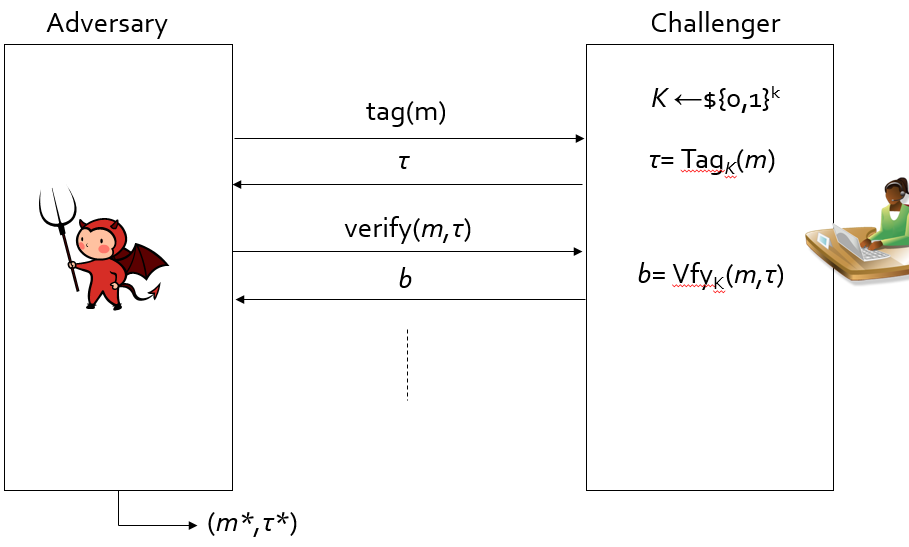
* Even when the adversary has seen many message/tag pairs (*m,τ*) computed using the Tag algorithm with the same key *K*.
* Even when the adversary gets to choose all the messages.

**Security for MAC**

* MAC tag *τ* must depend on every bit of the message *m*
  + Insecure example: Tag*K*(*m*) = *m0m1 … mk-1* ⊕ *K* (tag only depends on first *k* bits of message).
* The security definition also means that it must be hard to recover *K* given some pairs (*m,τ*) (otherwise attacker can recover *K*, and then forgery is trivial).
  + Insecure example: Tag*K*(*m*) = H(*m)* ⊕ *K* where *H* is a hash function with output size *k* bits.
* Length Extension Attacks
  + Insecure example: Tag*K*(*m*) = H(*K* || *m)* where *H* is a Merkle-Damgård hash function: insecure because, given Tag*K*(*m*), we can compute Tag*K*(pad(*m) || m’*) for any string *m’* by exploiting length extension property of iterated hashing.

Formalising security for MAC

* Game (adversary and challenger)
* Challenger choose random key
* Adversary has access to tag oracle/ verify oracle
* accessing the tag and verify oracles as much as it likes (wins if (m\*, t\*))



Weak unforgeability

(m\*) = new & VrfK(m\*,t\*) = 1

Strong unforgeabilitiy

(m\*, t\*) = new & VrfK(m\*,t\*) = 1

Win by coming up with a different valid tag *τ\** on a message *m* for which it queried its tag oracle and received a tag *τ*

adversary builds (*m*,*τ*) from one of its tag queries and its final output is (*m*,*τ*\*) where *τ ≠ τ\**

adversary would not win in the weak game (because it queried *m* to its tag oracle)

MAC scheme (Tag,Vrf) is **(W)UF-CMA or SUF-CMA secure** if there is **no** **efficient** adversary that wins the relevant game with a **significant probability**.

‘Efficient’ here = amt of time and memory used

Significant probability: can always guess the *t-*bit tag value, probability needs to be low

Means “not much better than these guessing adversaries do, averaged over random key *K* and adversary’s randomness.

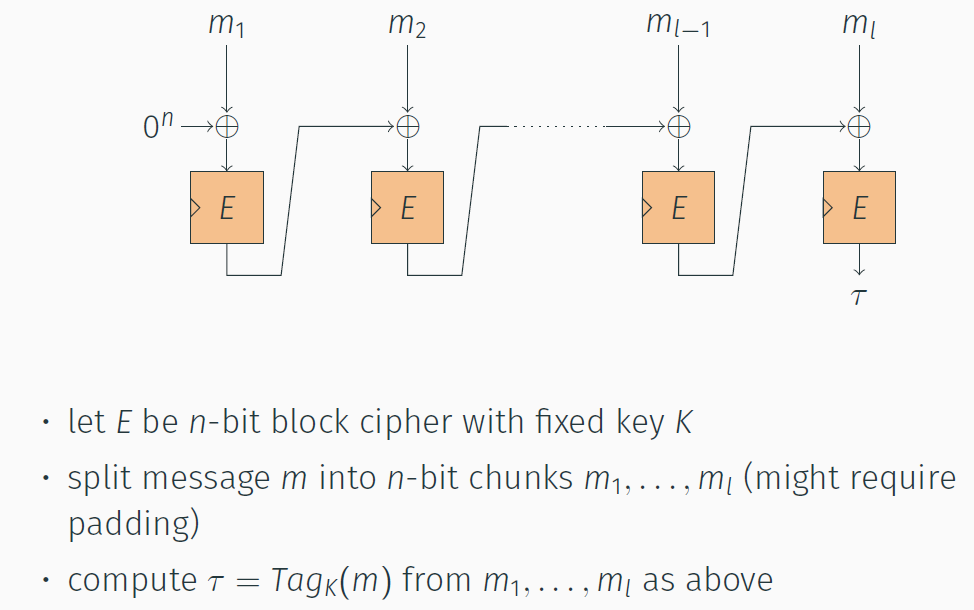
Key length: at least 2^128 bits, 80bits prevent offline exhaustive key guessing

Tag length: doesn’t need to be large, as the challenger can rate limit the adversary

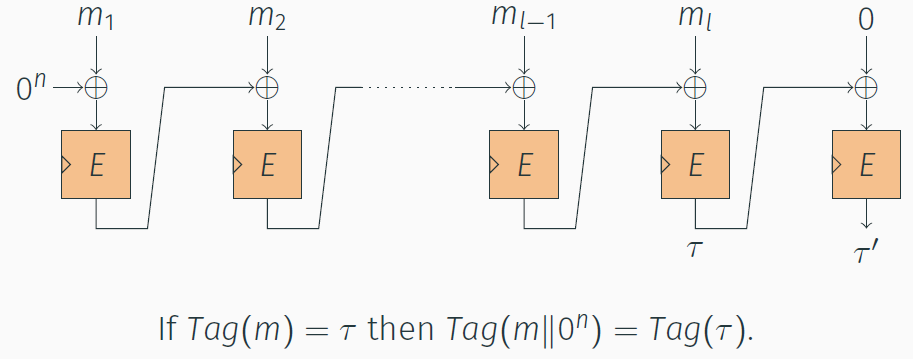
**Attacks MAC don’t prevent**

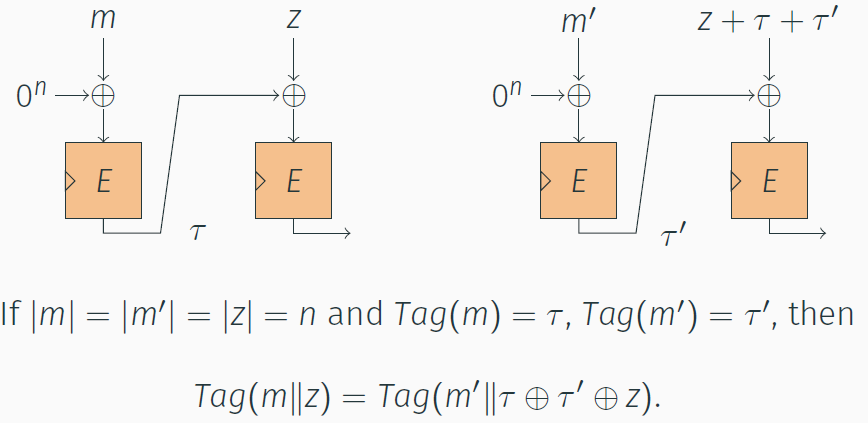
* integrity and data origin authentication
* MAC cannot detect message deletion, replay, reorder
* Same MAC key K used, then cannot prevent reflection attacks.
* **No confidentiality**

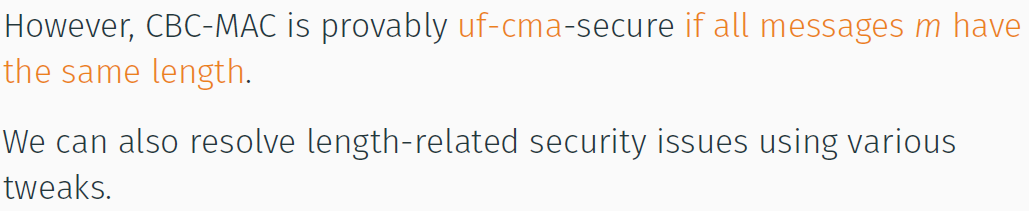
**CBC-MAC**

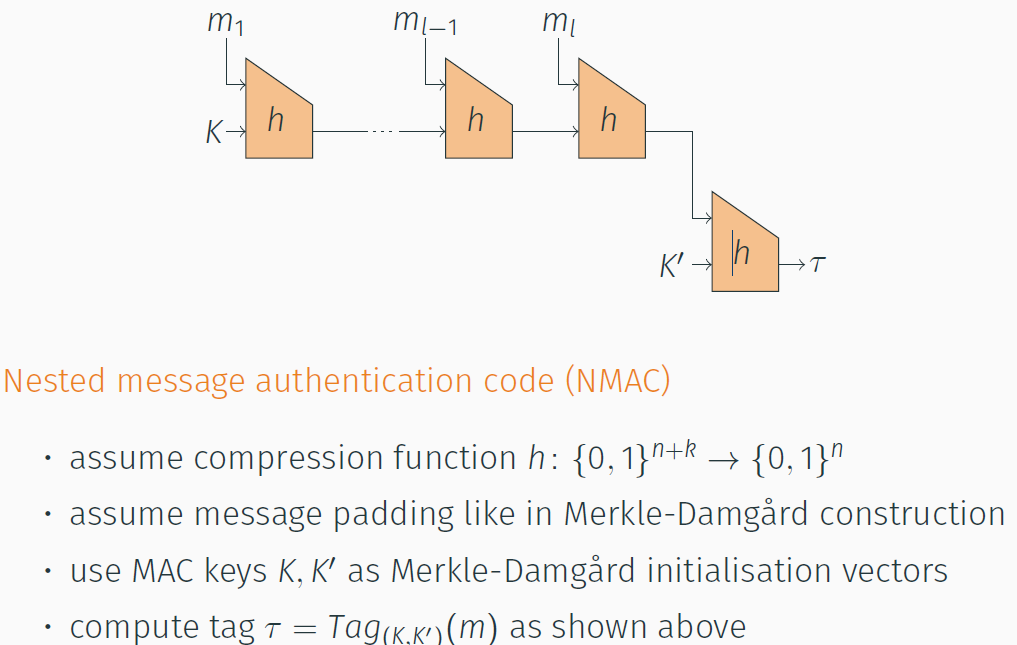


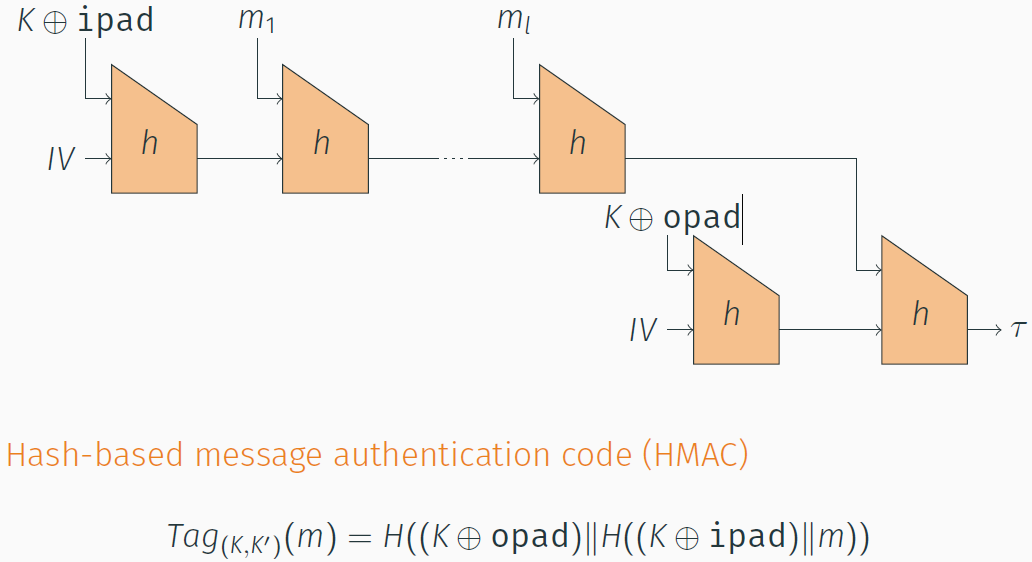
can get Toh prime by using 0

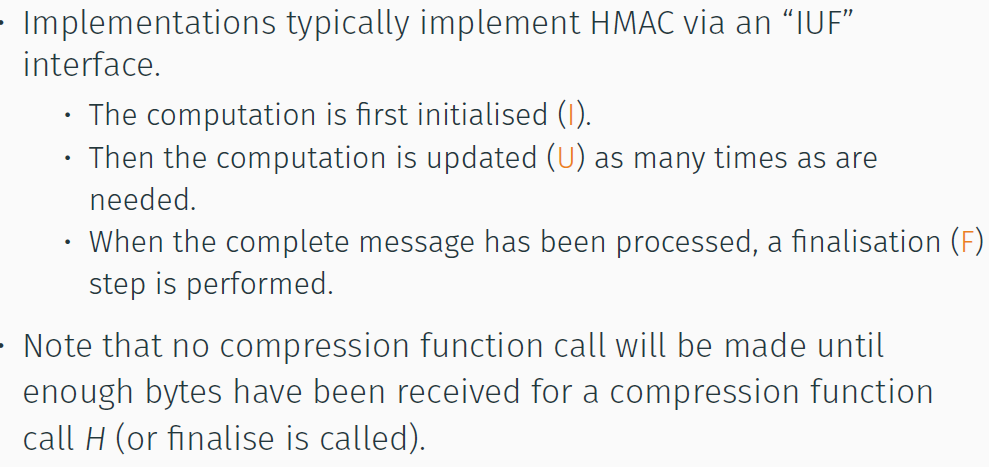
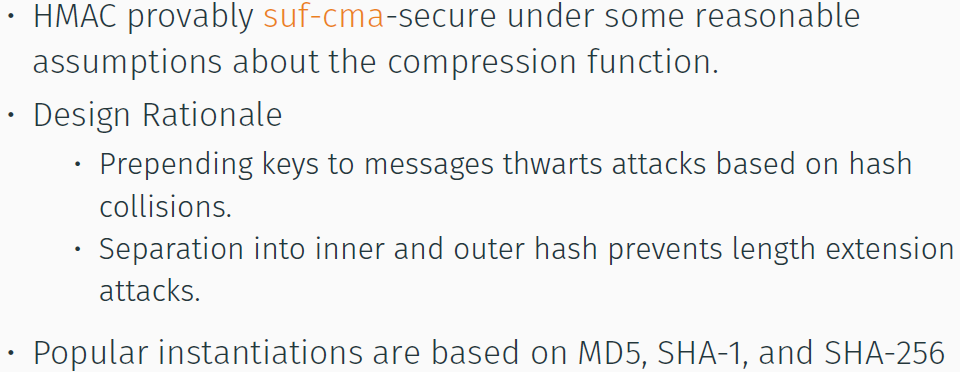




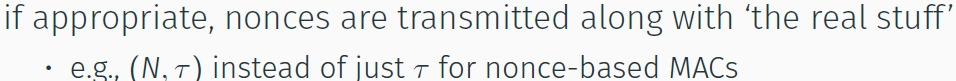


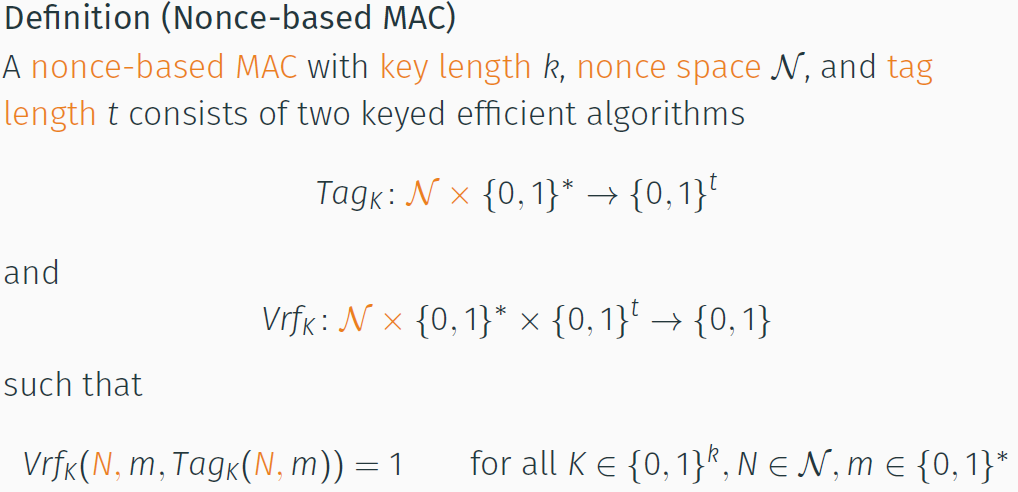


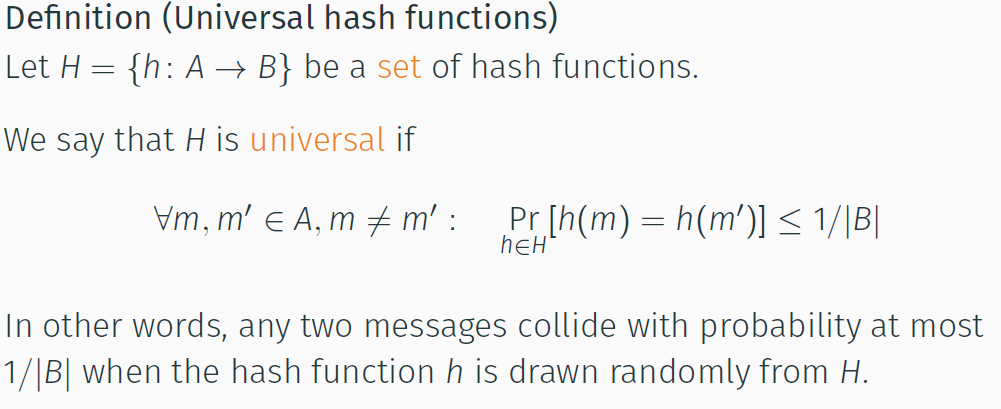


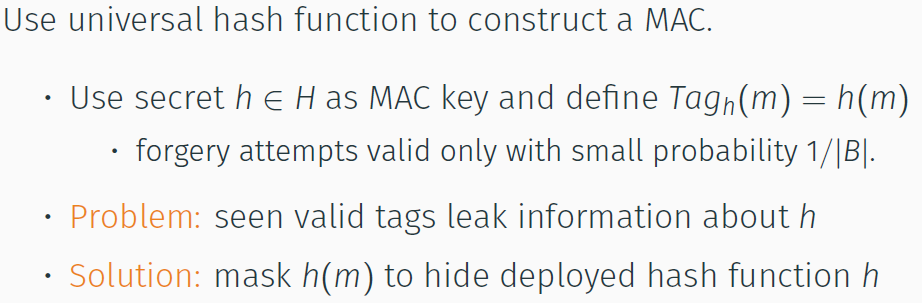


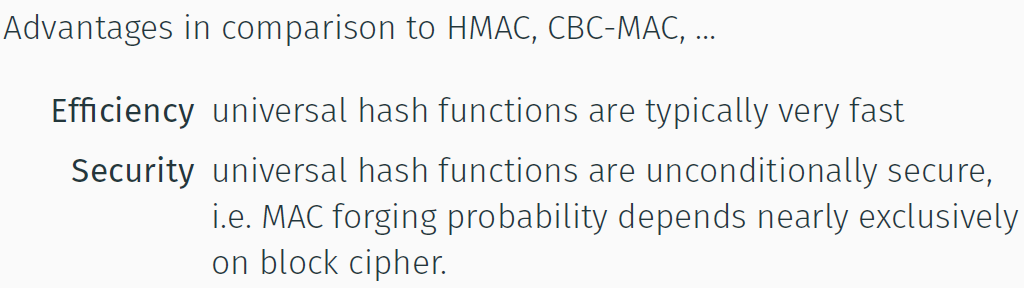
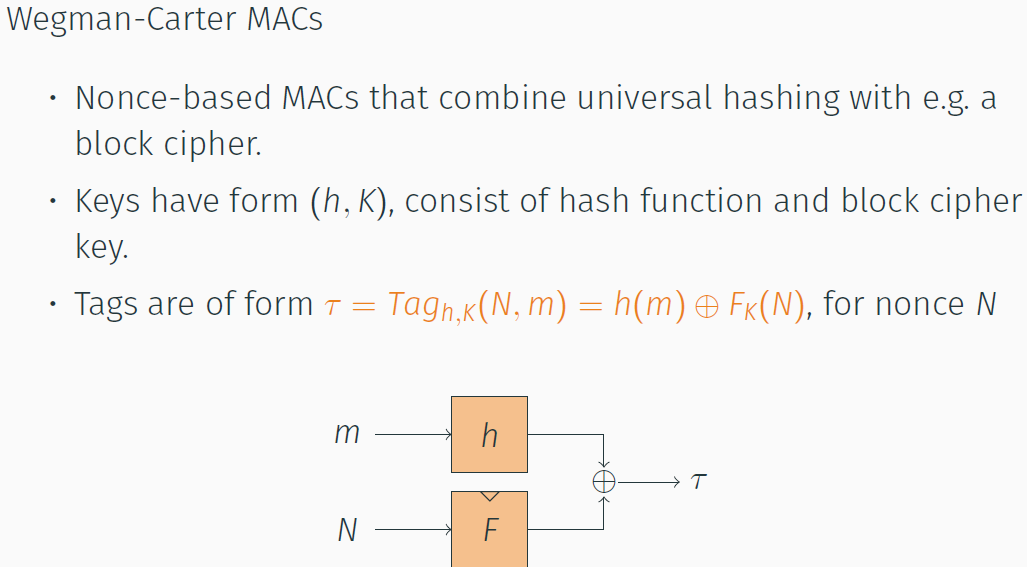
**Nonce based cryptosys**

* No need to be secret
* No security guarantee if reused (can use ctr)
* 

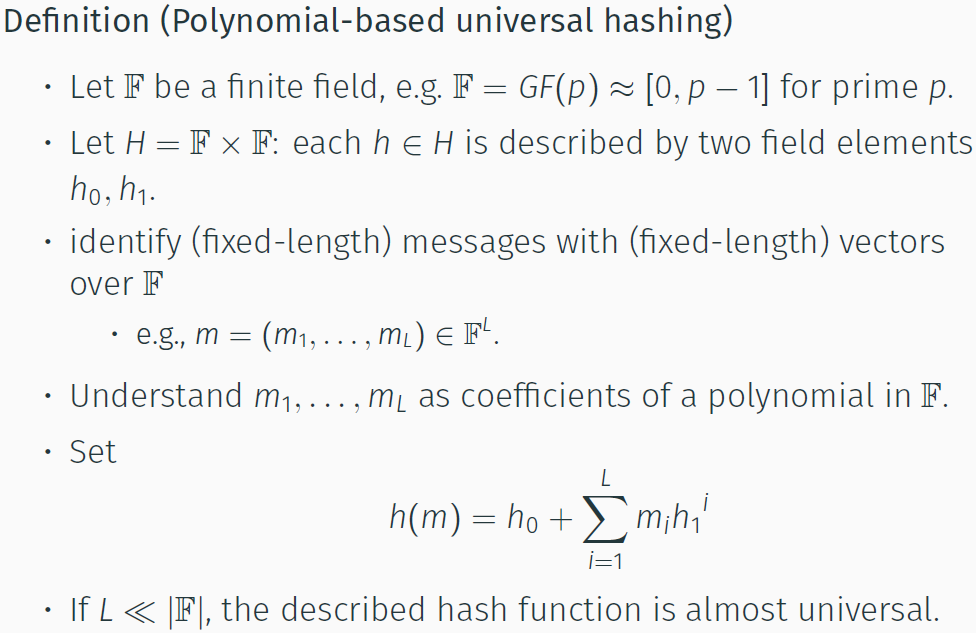




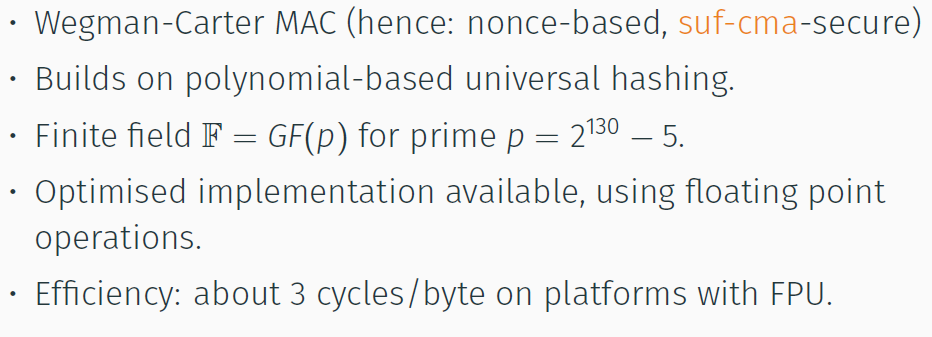




* **Encrypt only the Nonce (fast)**
* **Hash the whole message (hash is fast)**



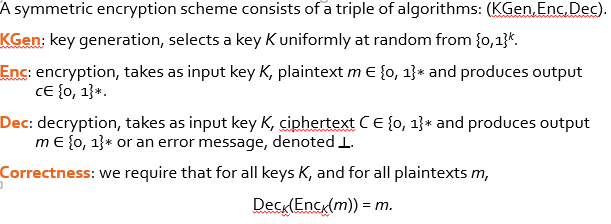
**Poly1305-AES**



**Authenticated Encryption (AE)**

* **Confidentiality + Integrity**

**Sym Enc**

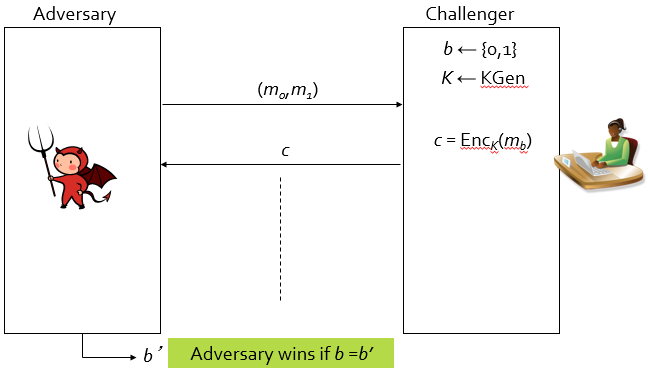


**Sym** Enc offer AE if:

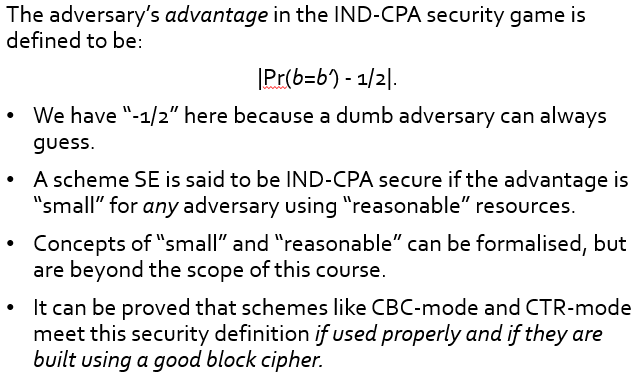
* **Chosen plaintext attacker** (nothing about plaintext, except length)
* **Cannot forge** ciphertext (free access to oracle)
* **Unconditional security**

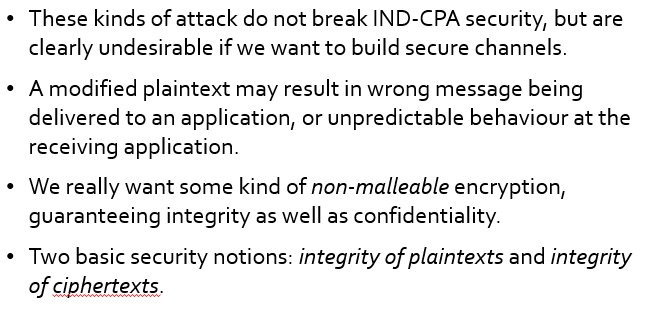
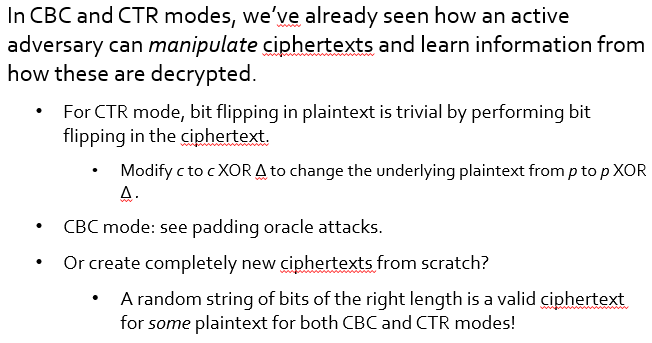
**IND (**indistinguishable) + **CPA (**Chosen plaintext atk)

* Repeated access to Left/Right encryption
* Adversary submits pairs of equal length plaintext (m0, m1) to the oracle
* **Gets back C,** encrypton of m**b** , where b is fixed but random bit
* **Adversary** estimate b, wins if correct

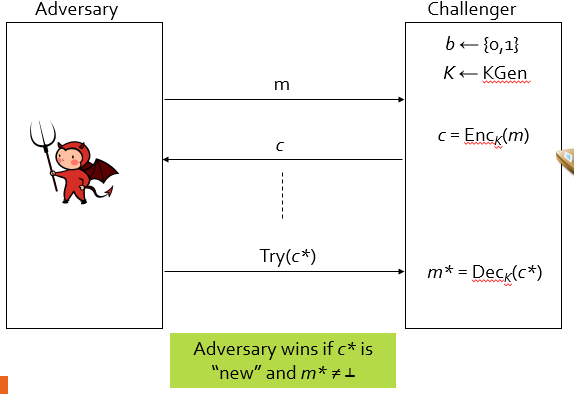
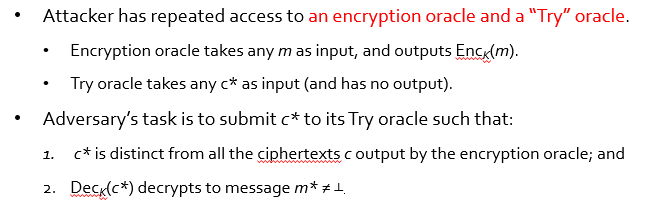


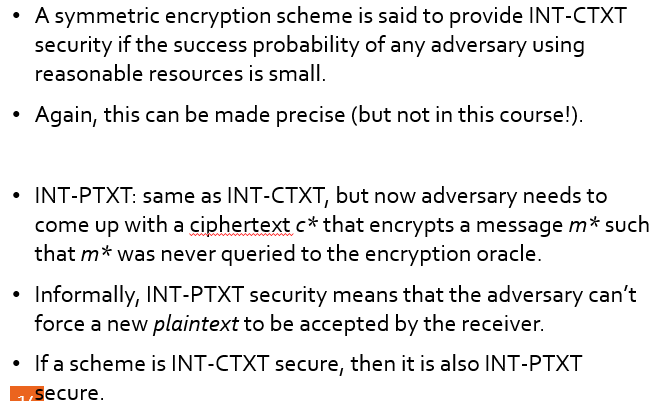
**K - > selected once at random**





**INT-CTXT**

****

****

**AE = IND-CPA + INT-CTXT**

Chosen ciphertext attack (guess plaintext)

* access to both an encryption oracle *and* a decryption oracle.
* IND-CCA security notion. Strong than IND-CPA

Attack model may arise in practice

* Might not learn full plaintext, but might get error during decryption

**IND-CCA**

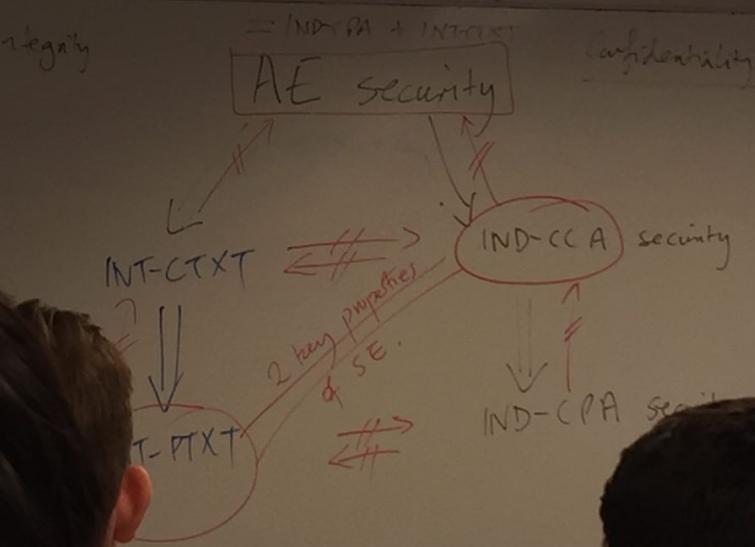


AE Security -> security against chosen cipher text attacks

**AE-Security implies IND—CCA implies IND-CPA**

**AE-Security implies INT-CTXT implies INT-PTXT**

**Symmetric Key Encryption = IND-CCA + INT-PTXT**



Does not protect reordering/deletion

**Generic for AE**

Options: E&M, MtE, EtM

**Encrypt & MAC**

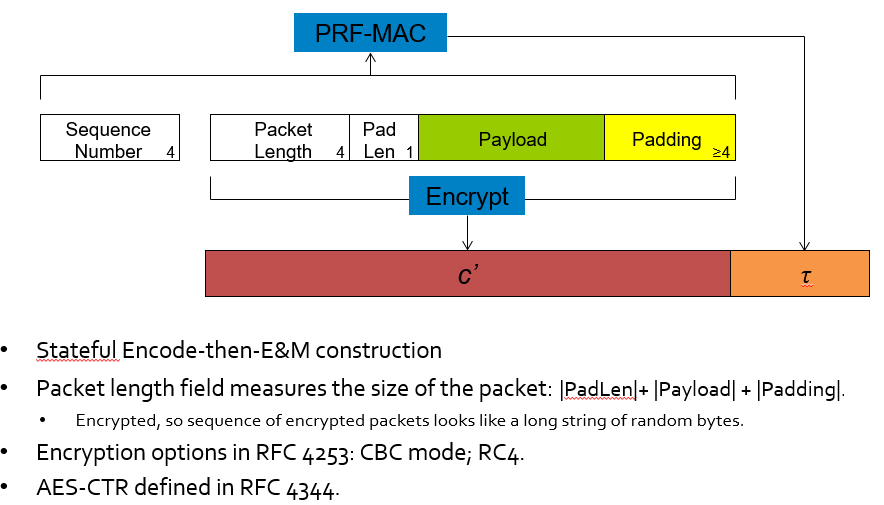
* compute *c’* ← Enc*KE*(*m*) and *τ* ← Tag*KM*(*m*) and output c = (*c’,τ*).
* used in SSH
* no AE security

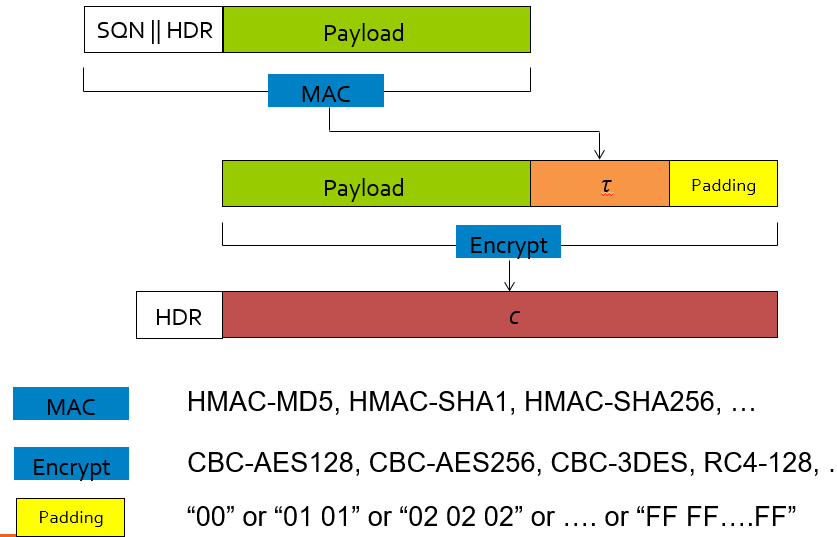
**MAC-then-Encrypt (MtE)**

* compute *τ* ← Tag*KM*(*m*) and output *c* = Enc*KE* (*m* || *τ*).
* used in TLS
* NO IND-CCA, therefore no AE
* But u can, if careful

**Encrypt-then-MAC (EtM)**

* compute *c’*← Enc*KE* (*m*) and *τ* ← Tag*KM* (*c’*) and output *c* = (*c’,τ*).
* used in IPsec ESP “enc + auth”

****

****

**Use EtM!**

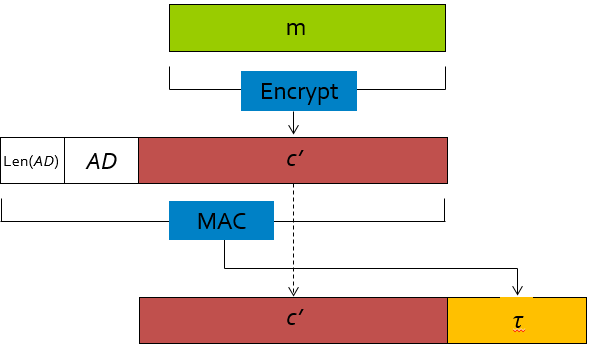
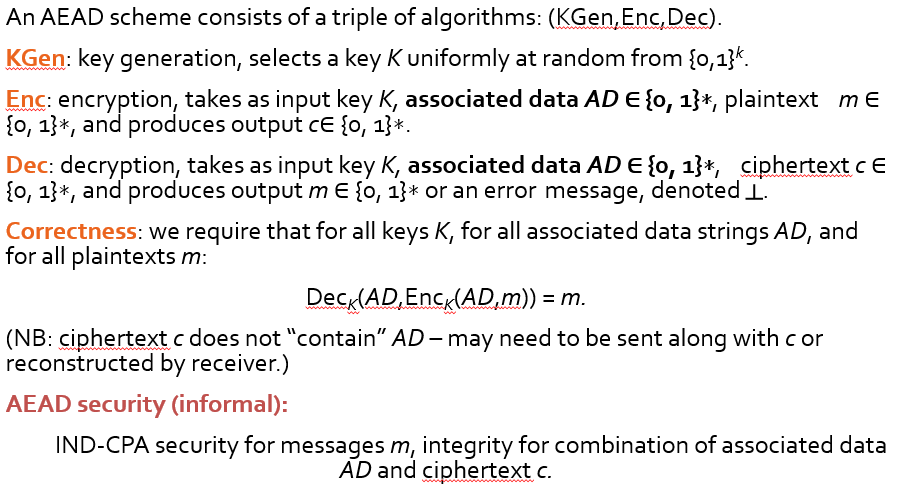
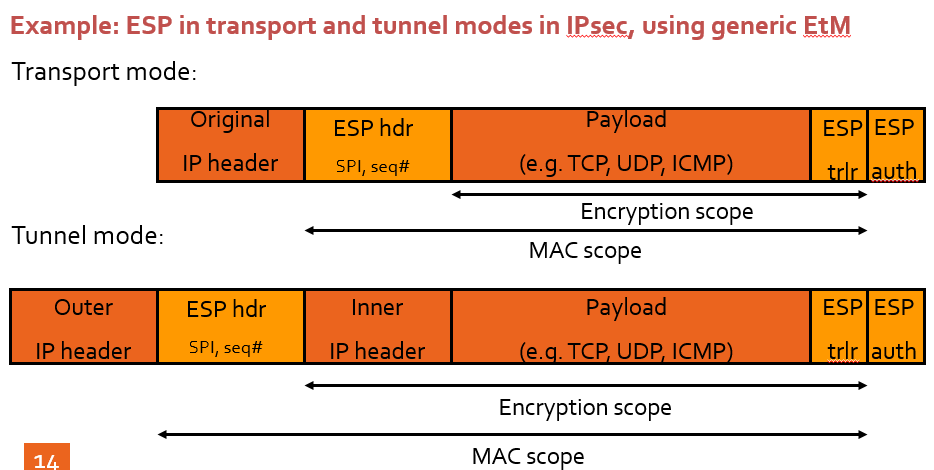
* EtM: compute *c’*← Enc*KE* (*m*) and *τ* ← Tag*KM* (*c’*) and output *c* = (*c’,τ*).
* AE security assuming encryption is IND-CPA secure and MAC is SUF-CMA secure.
* Intuition: MACing the ciphertext *c*’ provides ciphertext integrity; IND-CPA security of encryption carries over to the composition.
* Plus point: check MAC on ciphertext, don’t even decrypt if it fails; no temptation for programmer to “use the plaintext anyway” if MAC fails.

**AEAD (Authenticated Encryption, with Associated Data)**

**Wants to protected Associated data as well.**

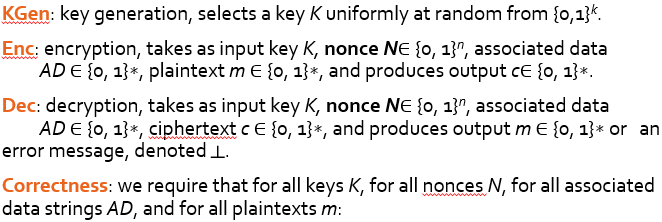
**Input requires AD and K**

****

****

**NAEAD ( Nonce based Authenticated Encryption, with Associated Data)**

-random is hard, use incremental counter

****

**Security**

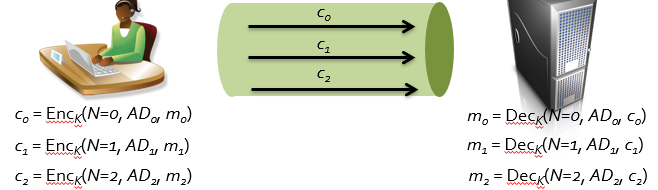
**IND-CPA + INT-CTXT + not repeating nonce**

**IND-CPA:**

* Adversary never repeats *N* (but is otherwise free to set its values).
* This very conservatively reflects the idea that a sender will always make sure to use a fresh value of the nonce each time Enc is called.

**INT-CTXT:**

* Adversary is permitted to repeat *N.*
* This reflects the idea that is some AEAD applications, *N* will be sent along with *c*, and then adversary can set *N* to any value it pleases (including with repeats).

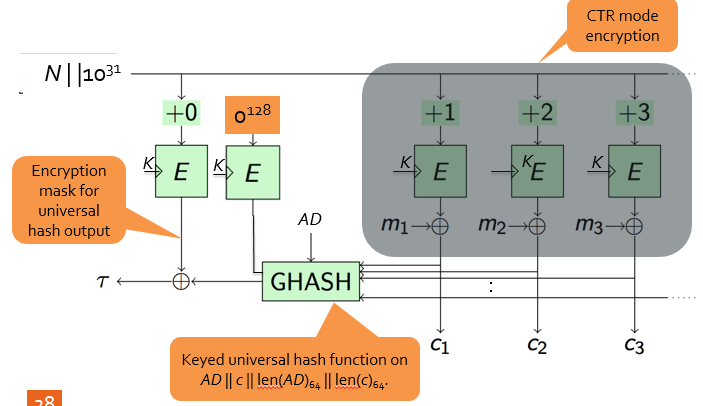


**If adversary Delete? Nonce DIFF, m = invalid**

**Reflection attack**

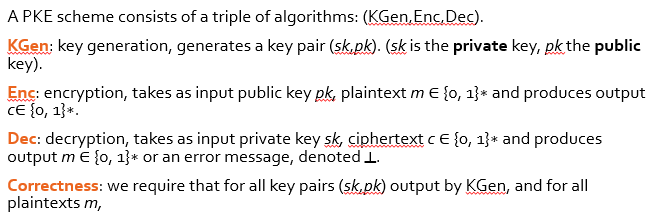
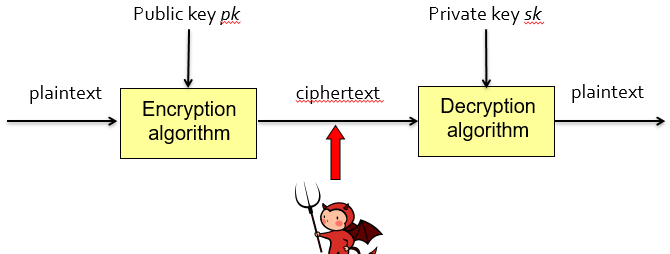
* **Use Bob message and replay at Bob**
* **Key separation (use different key for each direction to defeat)**
* **Direction indicator**
* **Results in Data orgin**

**GCM**

**RSA**

**PKE (Asymmetric Encryption)**

**Private Key = decryption  
Public Key = encryption**

****

**KeyGen = (sk, pk)**

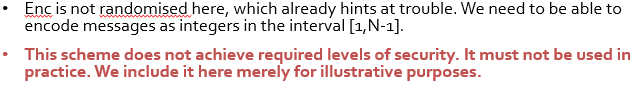
****

**KGen = randomized = so everyone has a different set of keys**

**Encryption = randomized (for security)**

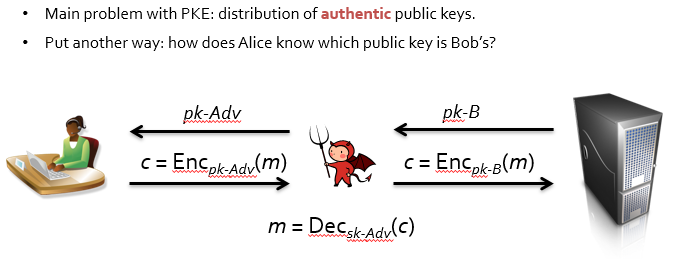
****

**Random primes = how?**

****

**Often used to transport symmetric key**

**Good for short messages (PIN encryption in EMV)**

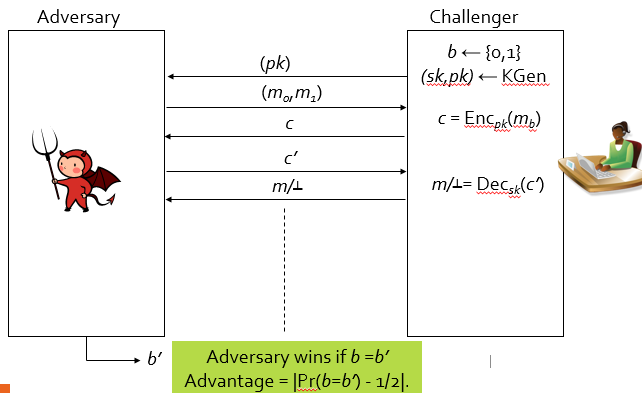
****

* **Use digital cert （**for key management）

Security for PKE:  
- exhaustive key search  
- exhaustive key gen  
- must be many possible key pairs  
(2^k)k

Breaking textbook RSA = factoring k-bit integer of product of 2 k/2 primes

FORMALISED version (IND-CCA)



**A PKE scheme (KGen, Enc, Dec) is said to be IND-CCA secure if the advantage is “small” for *any* adversary using “reasonable” resources.**

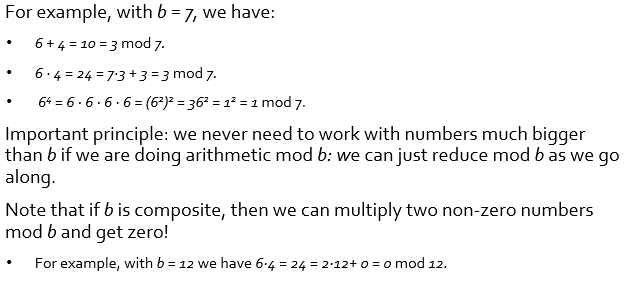
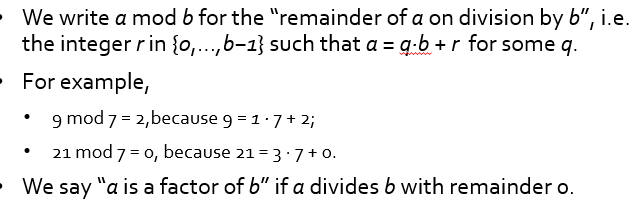
**IND-CPA (same, but remove decryption oracle)**

**IND-CCA implies IND-CPA**

**If deterministic ENC algo, PKE != IND**

**Textbook RSA = deterministic ENC algo**

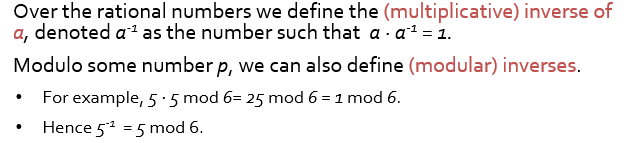
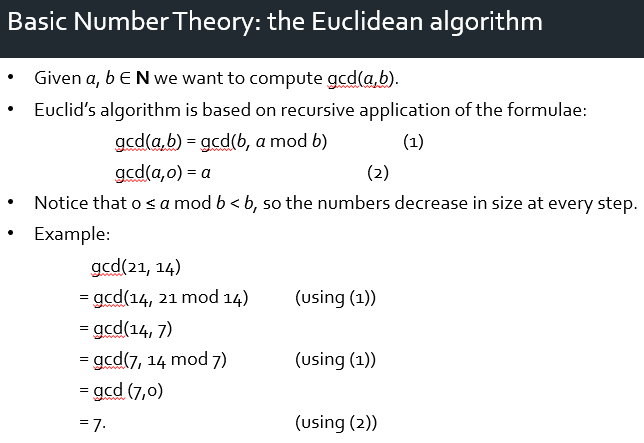
**Basic Number Theory**

****

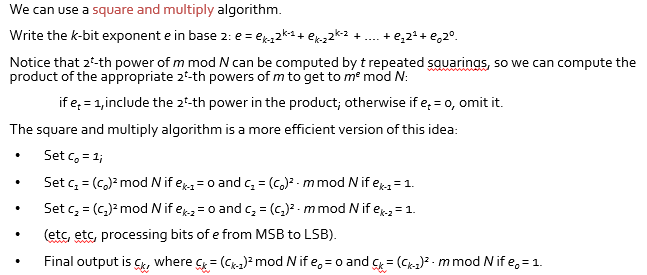
**Prime and GCD**

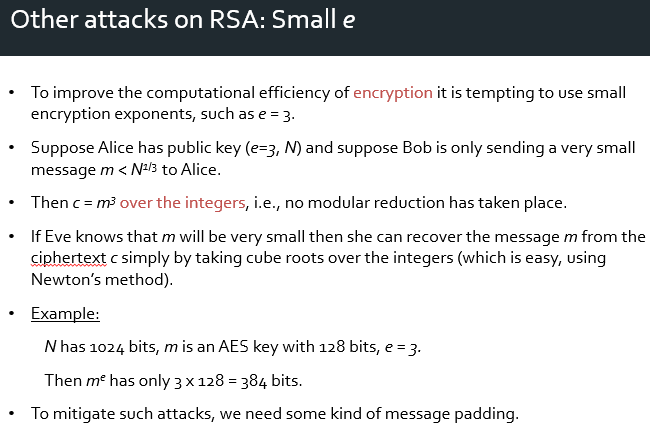
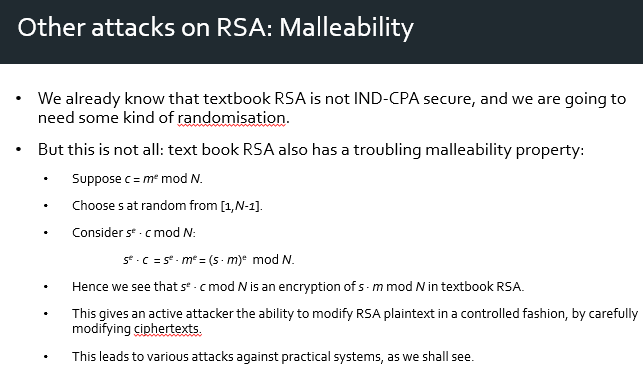
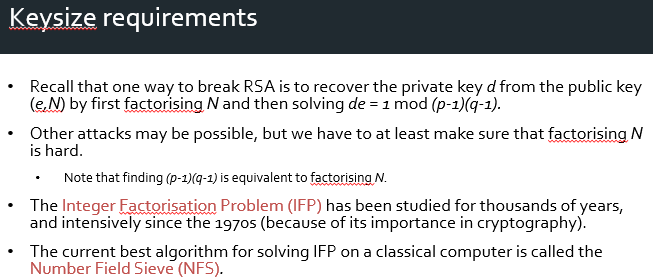
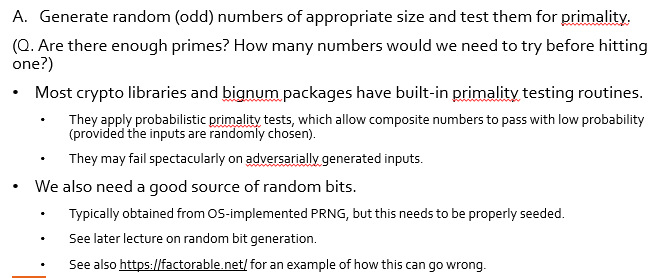
**2 number = relatively prime, GCD = 1**

**Euclidean algo**

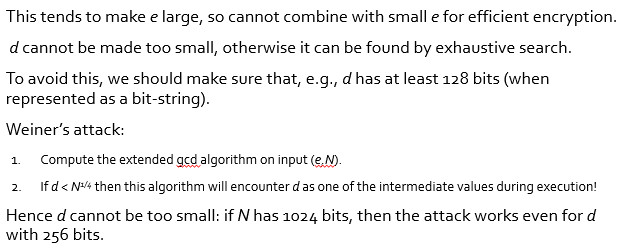
****

**MOD too slow, use square and multiply**

****

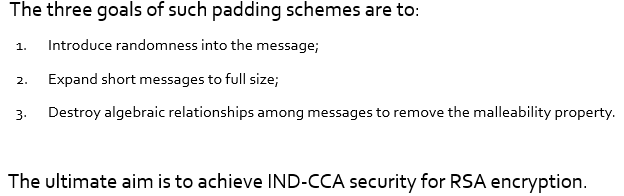
**Generating LARGE prime for RSA**

**SMALL E = if cant mod N , then just (e/cube) root to get m**

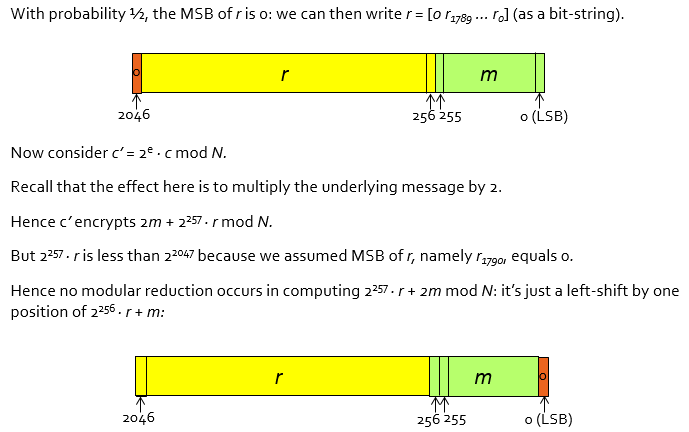
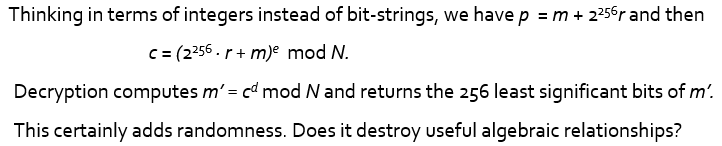
****

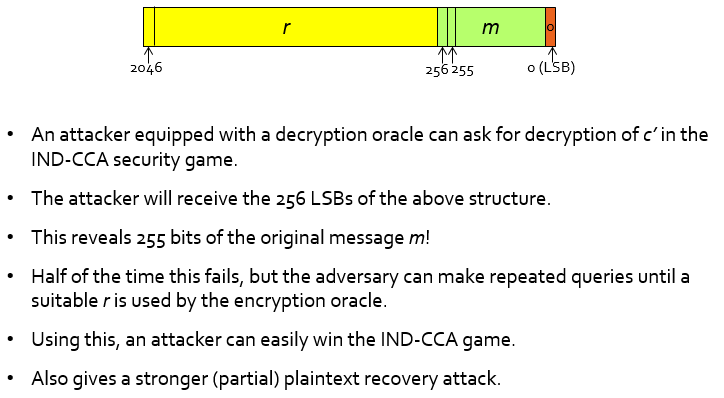
**SMALL d = recover d from N and e.**

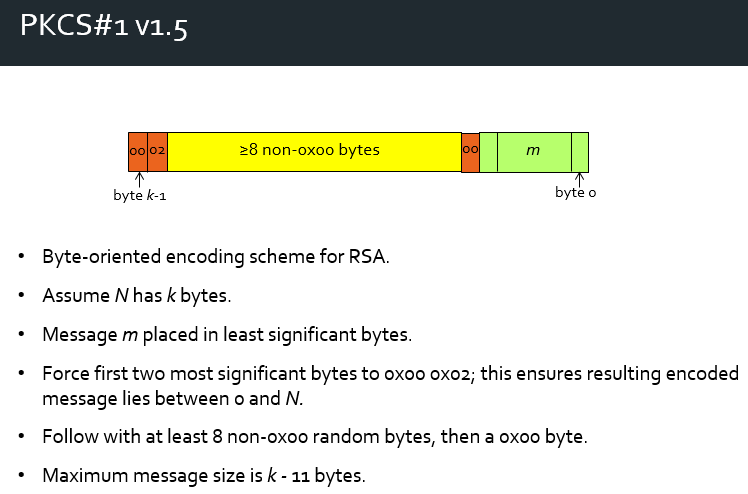
**Padding for RSA**

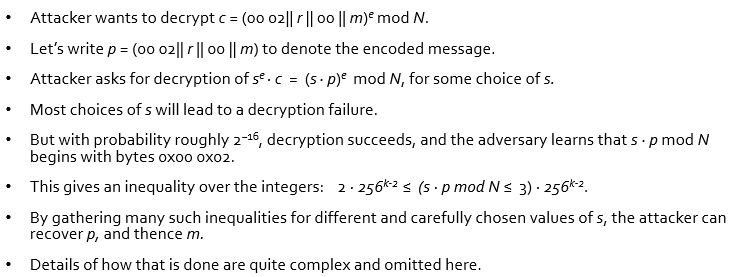
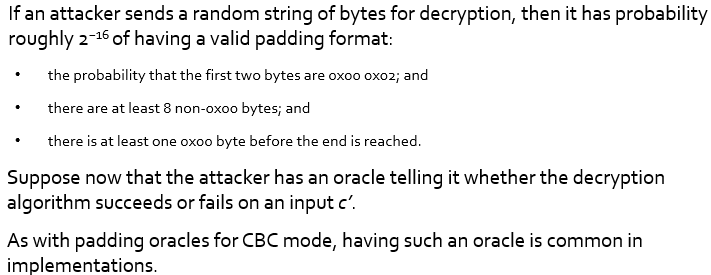
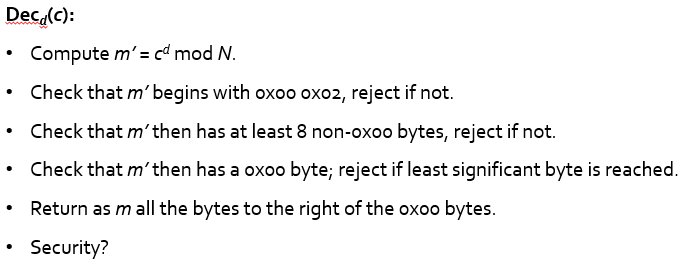
****

**PKCS#1 v1.5 = bad**

**IND-CCA security = RSA-OAEP. = good**

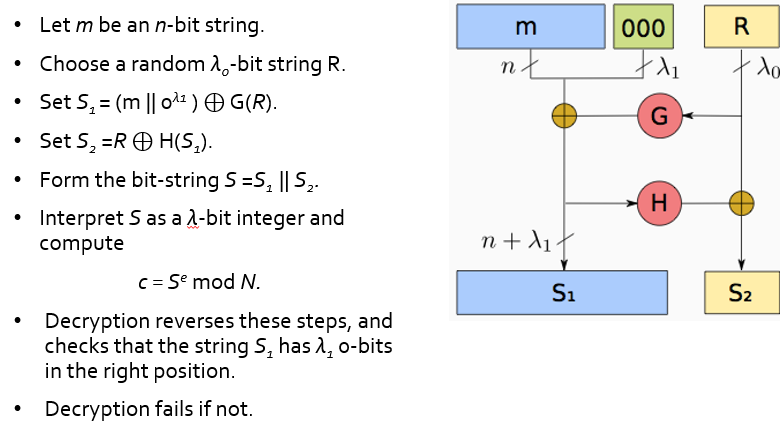
****

****

****

**around 220 in its first version, improved now**

patched the attack, instead of the structure

RSA-OAEP  


Rationale

* Randomized, full length
* *S1* and *S2* are both outputs of hash functions, the bit-string *S* looks random and algebraic relationships
* a decryption oracle and you send it a random value mod *N*, then the decryption process will fail with high probability, so the decryption oracle essentially becomes useless.

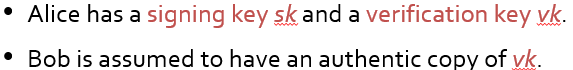
**RSA-OEAP can be proven to be IND-CCA secure,**

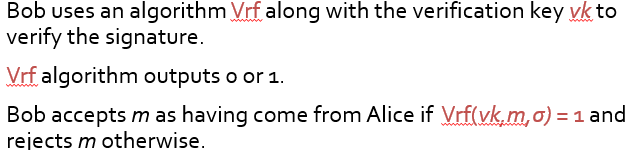
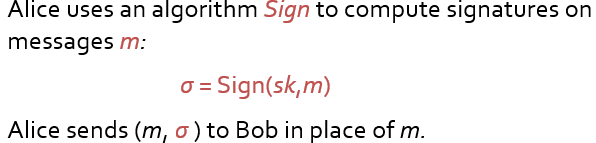
* **strong assumptions on G and H,**
* **under a strong number theoretic assumption (much stronger than assuming that factoring is hard)**

**Exam notes (under why and rationale, not implementation)**

**Many PKE, RSA = 1st that works**

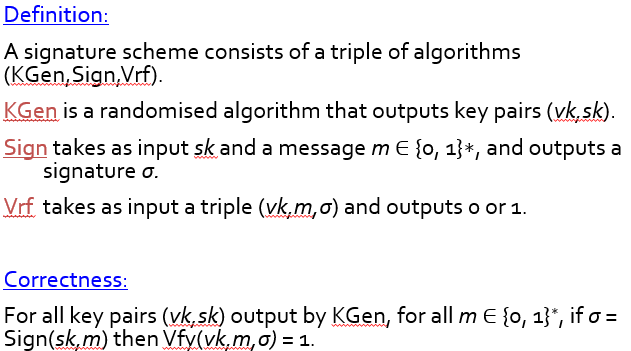
* large-scale quantum = can break RSASignature
* Wants compromise the **integrity**
* **MAC =** need Sym key





Hard to find m,*σ* such that Vfy(*vk,m,σ) =* 1.

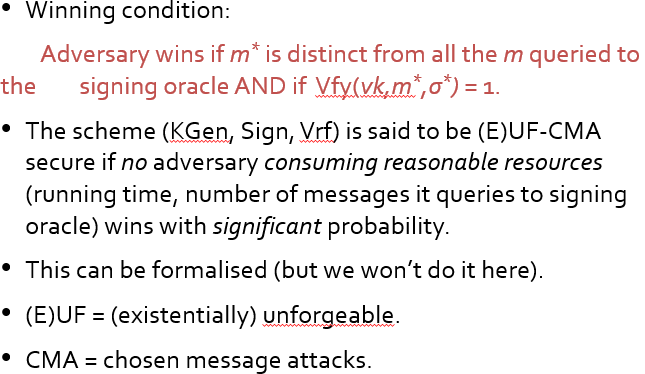
Hard to forge signature using *vk*



Security Game

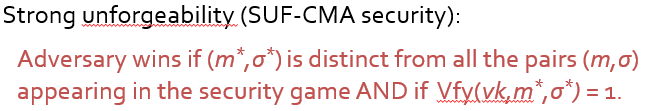
* Challenger generates a key pair (*vk,sk*) by running KGen.
* Challenger gives *vk* to adversary.
* Adversary runs, with access to a signing oracle: adversary sends *m* and gets back *σ =* Sign(*sk,m*).
* (No access to a verification oracle, cf. MAC security.) \*Already have vk\*
* Adversary finally outputs (*m\*, σ\**).

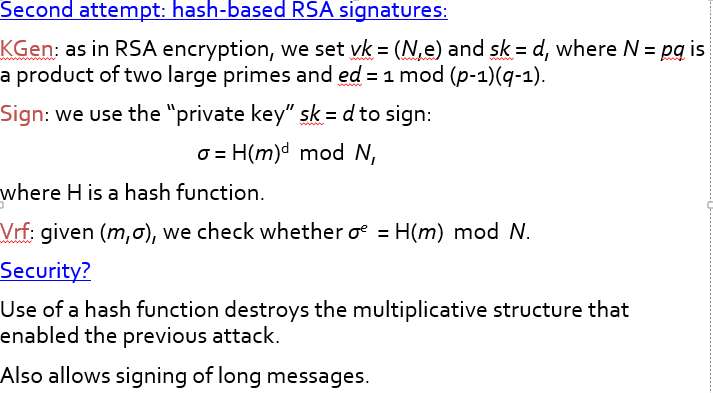
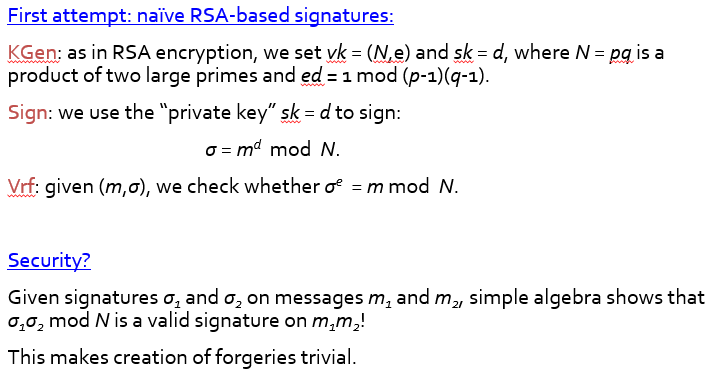
Winning condition:

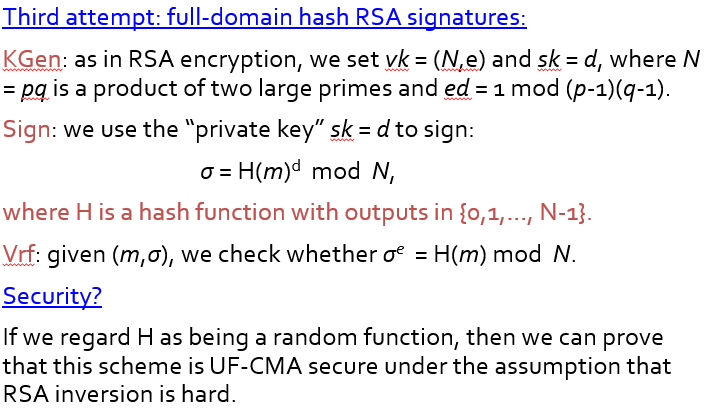
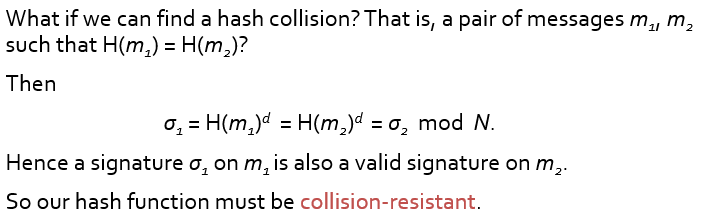
Adversary wins if *m\** is distinct from all the *m* queried to the signing oracle AND if Vfy(*vk,m\*,σ\*) =* 1

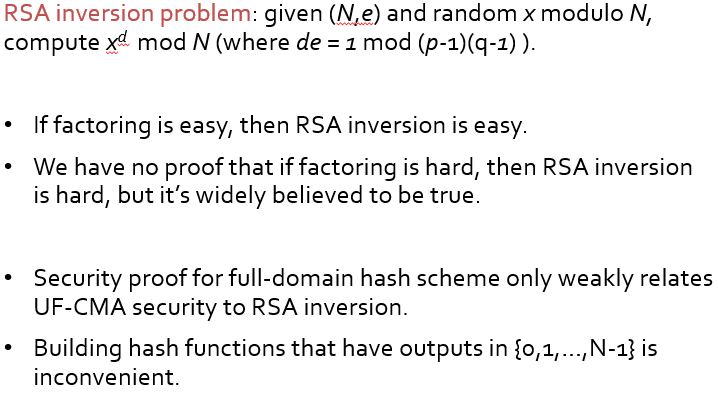
UF-CMA imples:

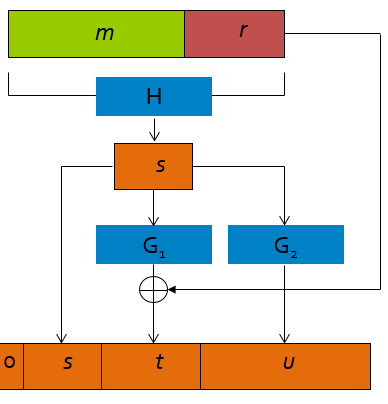
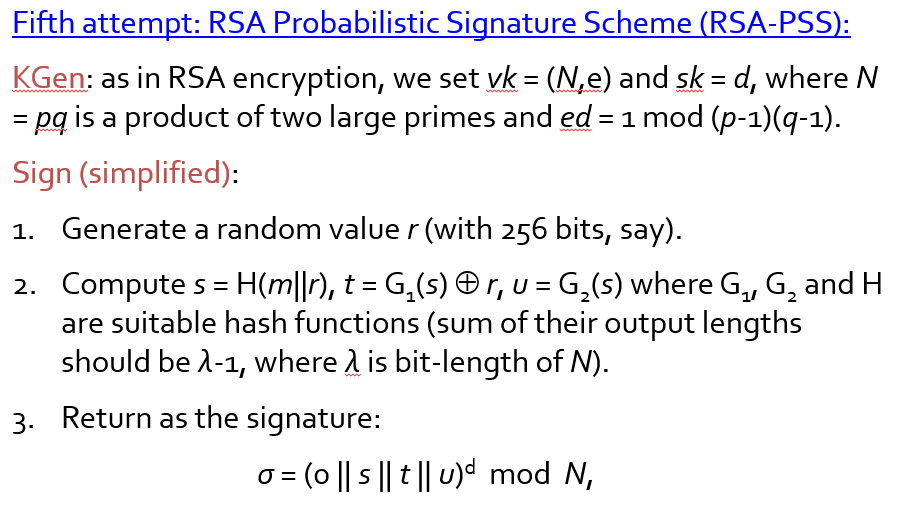
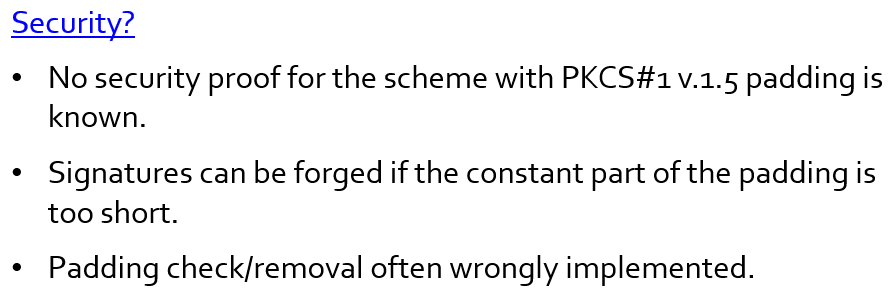
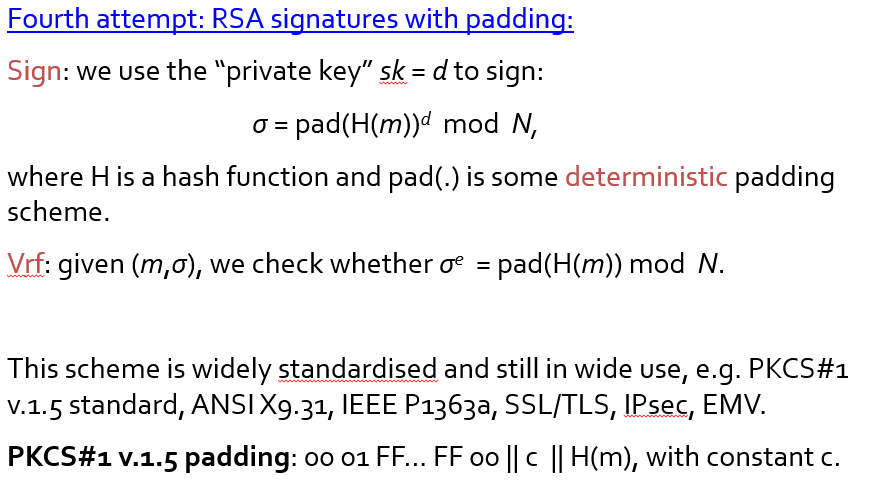
* Hard for adversary find sk from vk (can easily sign messages using sk)
* Hard for adversary to create a forgery given verification key vk
* Hard for the adversary to create a forgery when given access to signatures on random messages (controlled message already cannot)
* The legitimate owner of the key pair (*vk,sk*) cannot deny having created a signature *σ* on a message *m*. (non-repudiation) forgery not possible



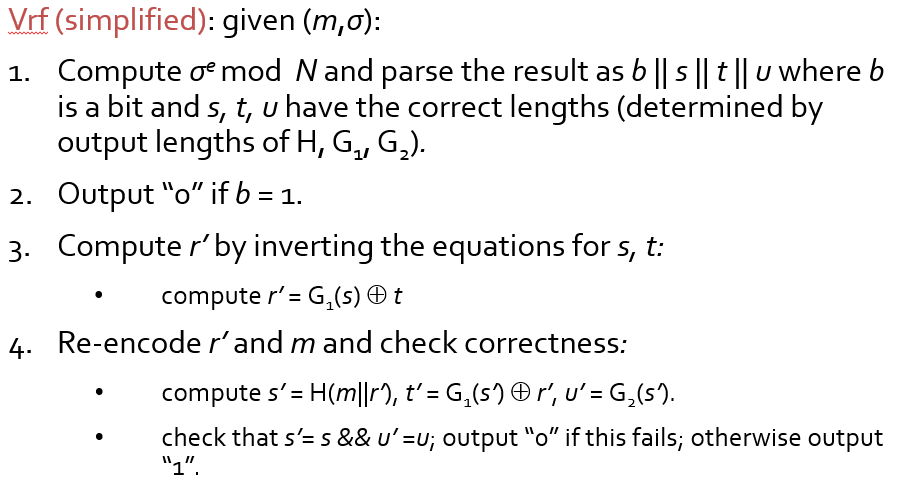


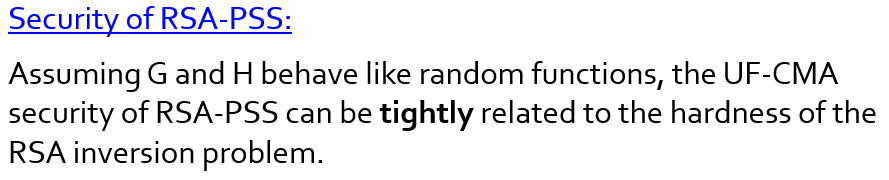






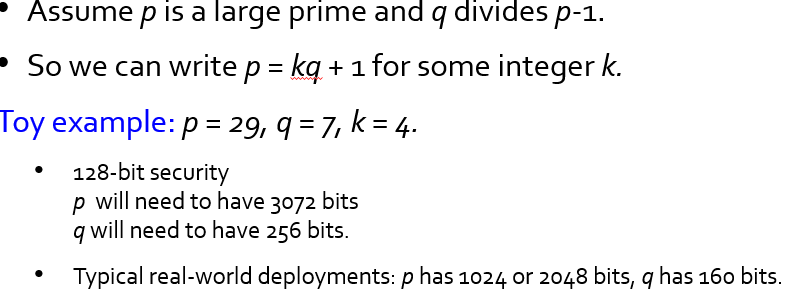
U = variable size (determine by s,t,0, N)



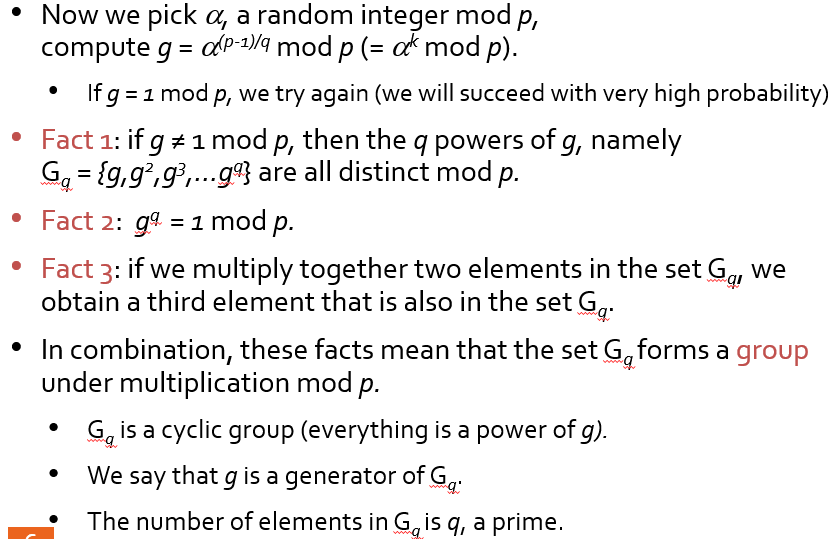
PSS = stronger and tighter than Full-domain hash  
(for UF-CMA secure)

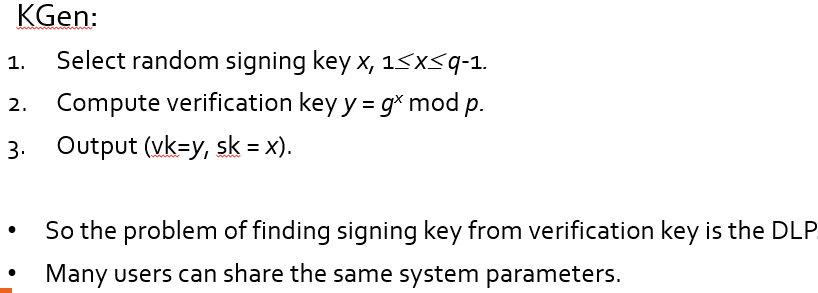
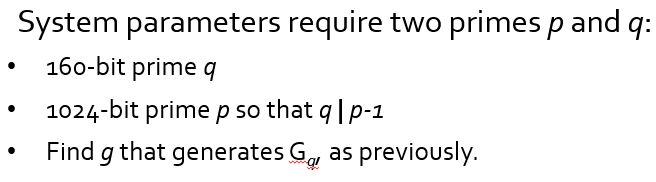
Discrete algo setting

* another platform for carrying our public key cryptography.
  + Digital sig, key exchange, and finally public key encryption.



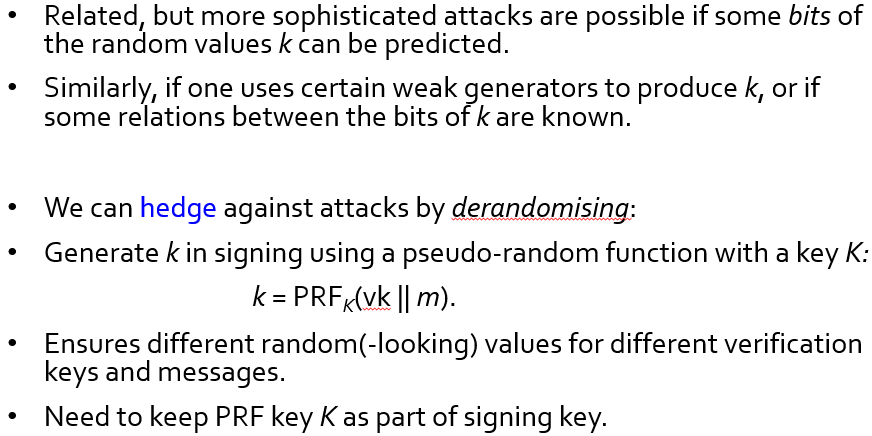
DISCRETE LOG SETTINGS



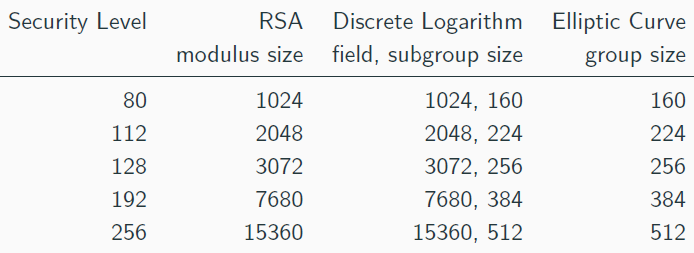
**DSA Setup**

**If random = same, key = same, private key = same, can get k, can get x (secret key) = broken.**

**Few bits of random k not random = also can work**

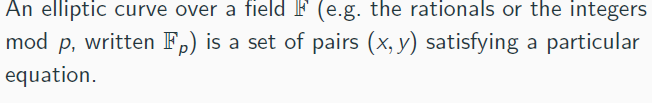


**Elliptical Curve**

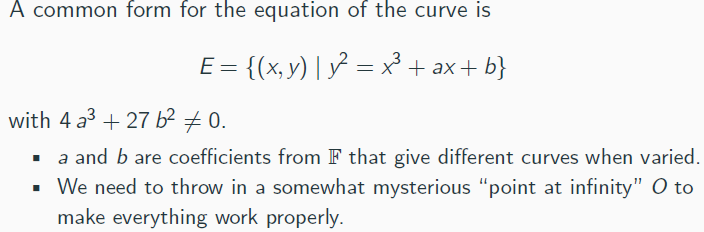
****

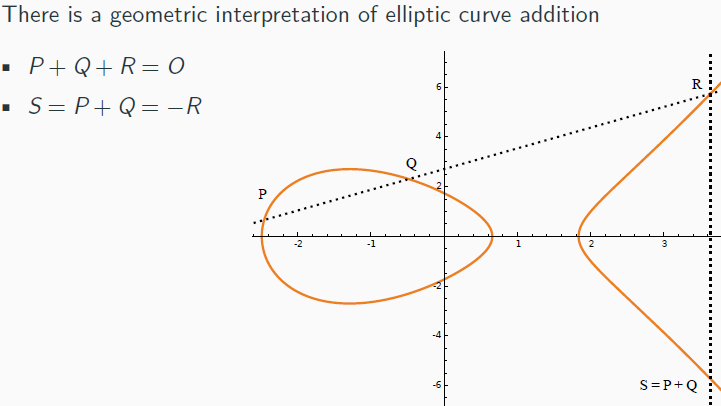
Moving to class of groups from EC = different hard problem

* ECDLP much harder than DLP
* Best algo = O(sqrt(q))
* Can use small params



EC = a field of(x,y) that satisfies equation





Finite field of 5 (possible set of values)

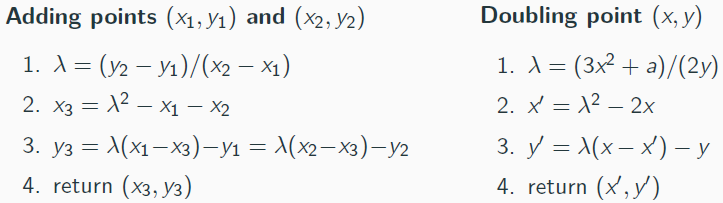
7 distinct points

Points can be added = for 3rd point

Anything + infinity = same as anything

P+Q+R = O

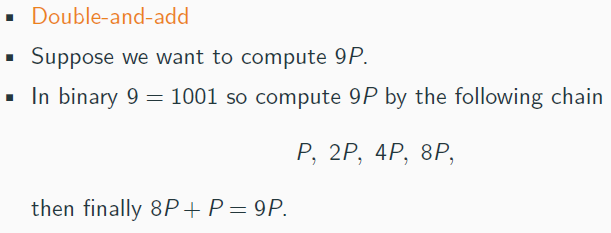
S=P+Q = -R

Adding points/ Doubling points

Scalar multiplication (using double and add)



Computing Scalar Multiples (double-and-add)

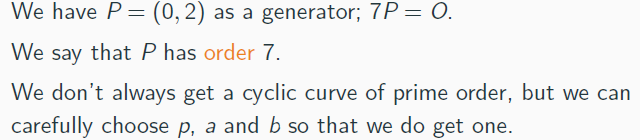


(like square and multiply in others)

Elliptic Curve Groups

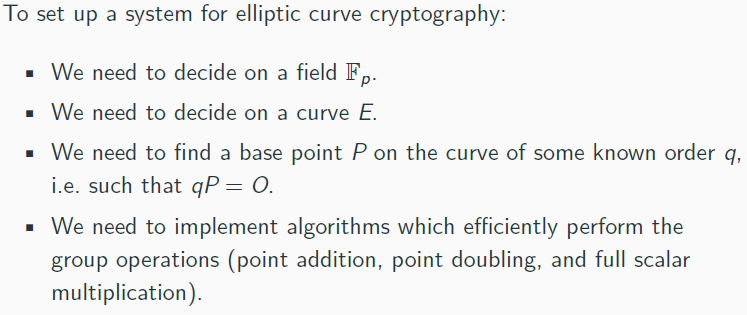
Field 5 = cyclic order of 7 (prime, no of points)

* P has order 7 (need add itself 7 to hit infinity)



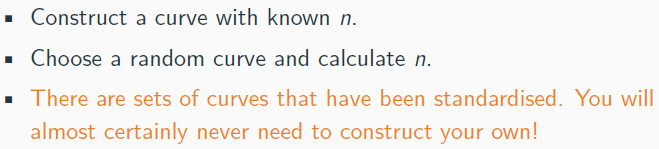
Finding t given Q and P

EC Setup

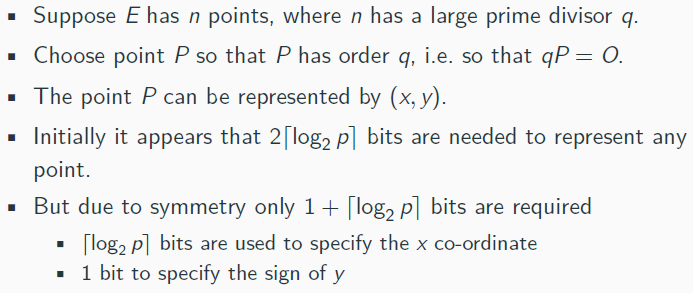


Choosing EC



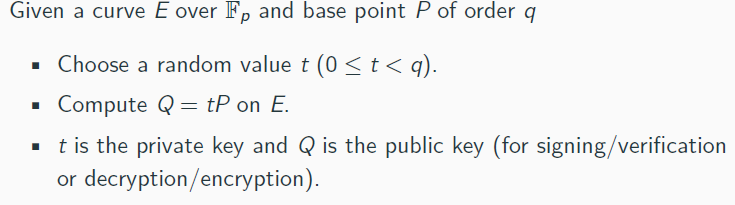


Points Compression: (reduce space to store points)



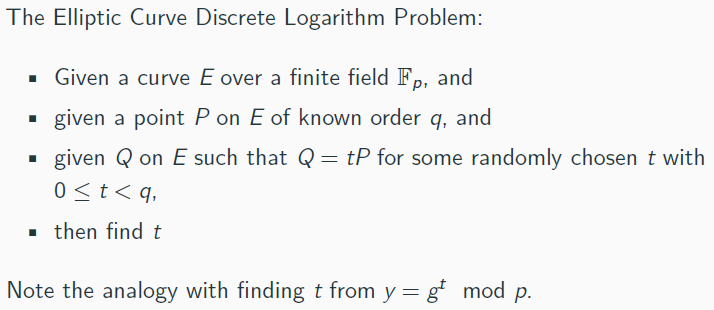
ECDSA

Q = tP on E   
- t = private key



DLP = find x

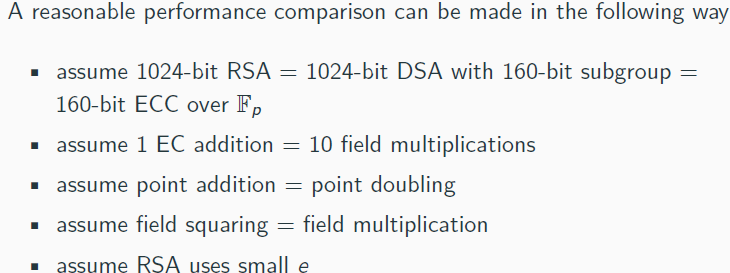
ECDLP = find t

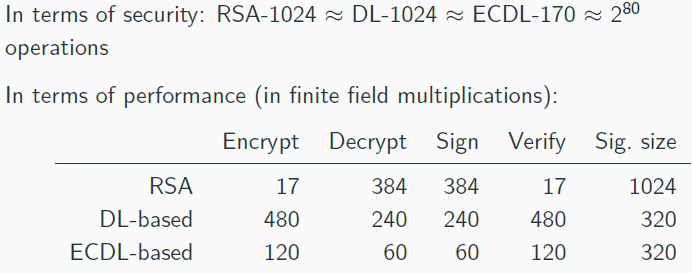


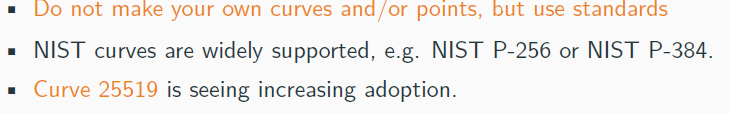
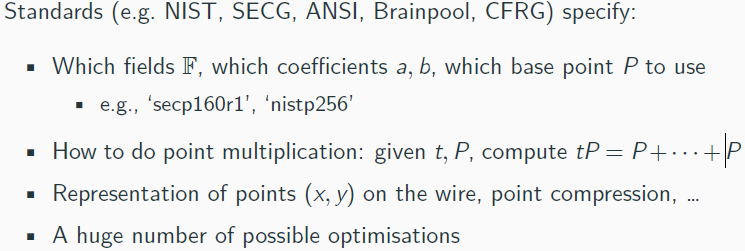
Security = no sub-exponential

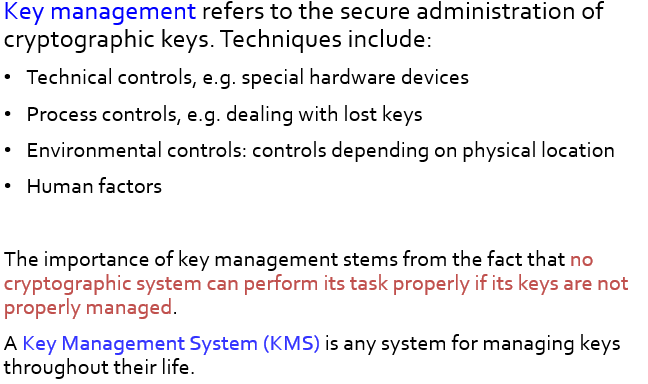
Best algo = square root time



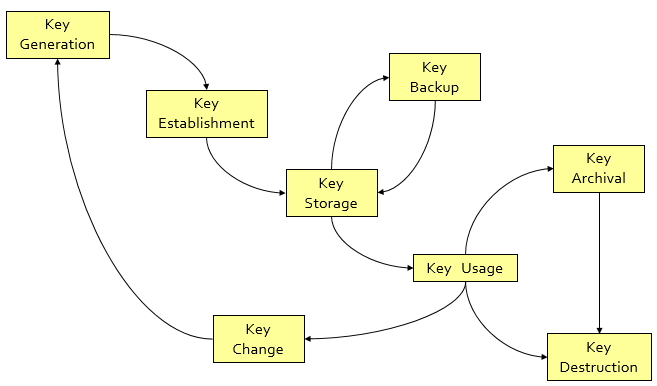
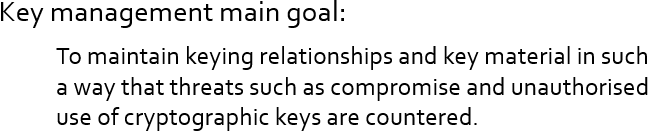
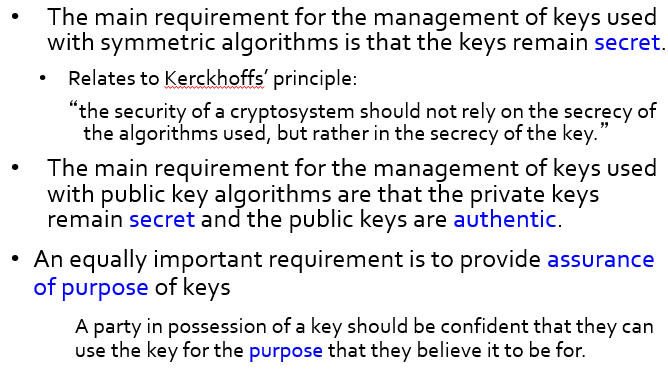
ECDSA = EC version of DSA

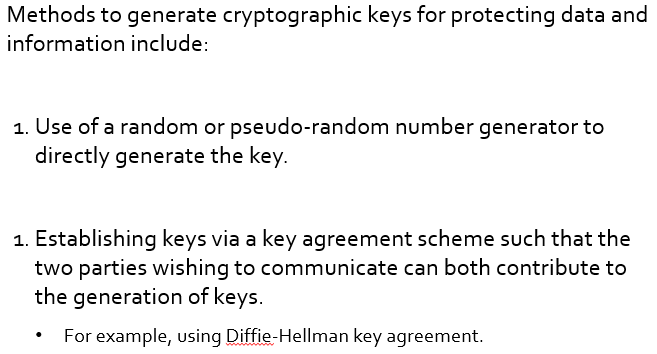
Performance

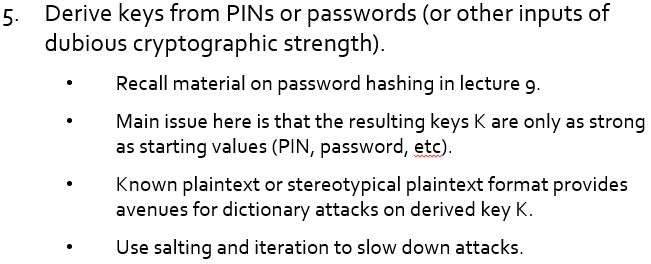
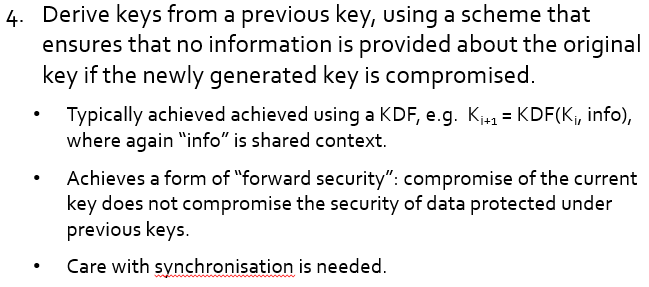
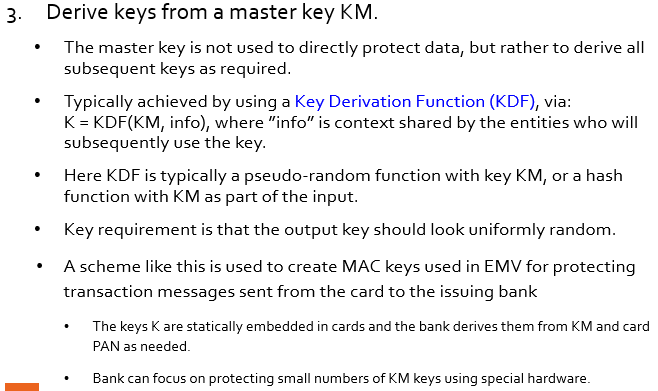


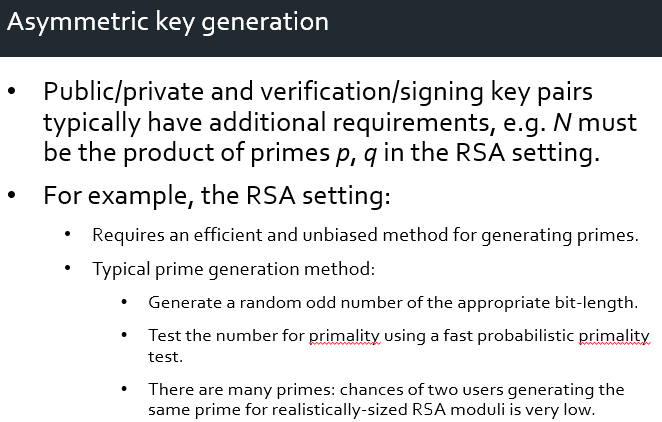


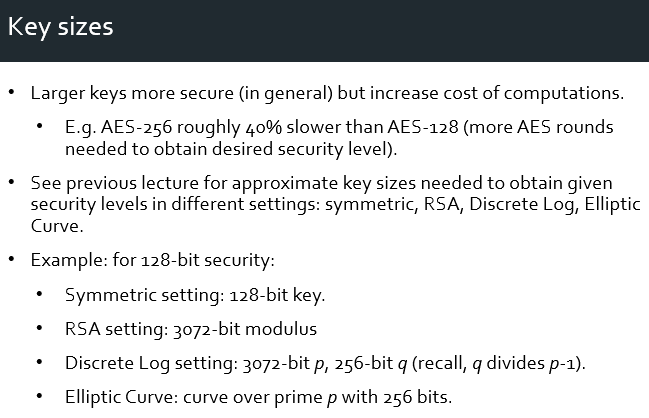
**3 main requirements of key management:**

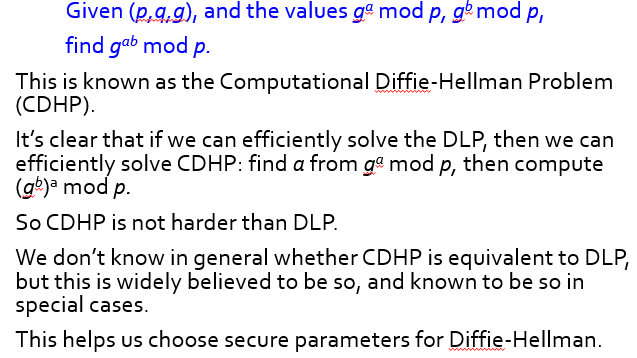
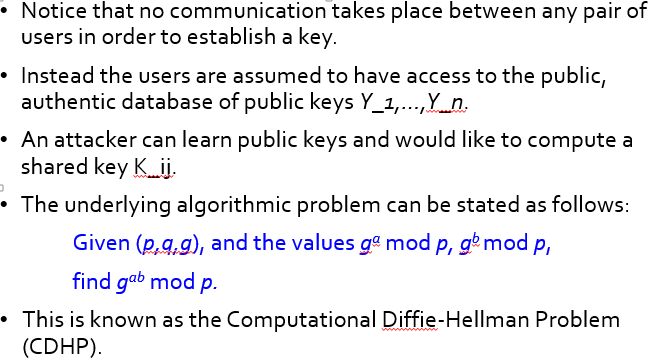
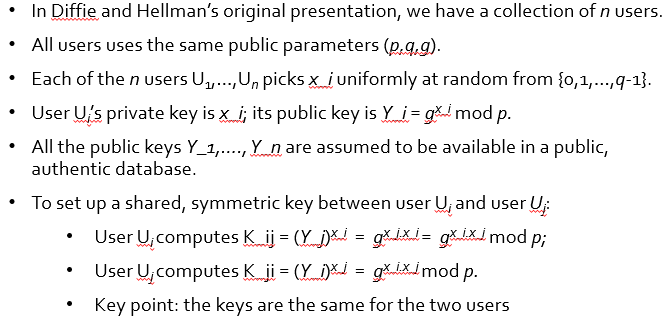
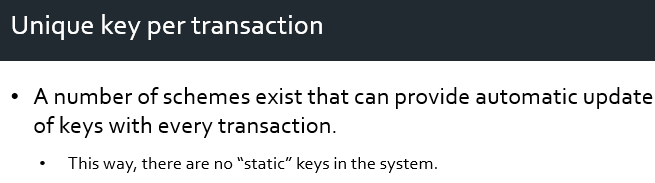
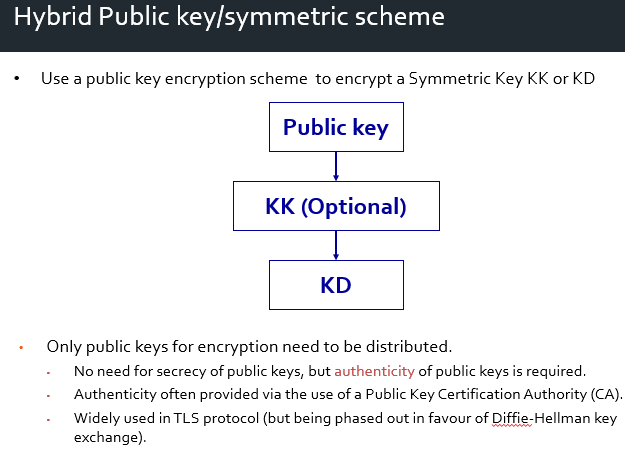
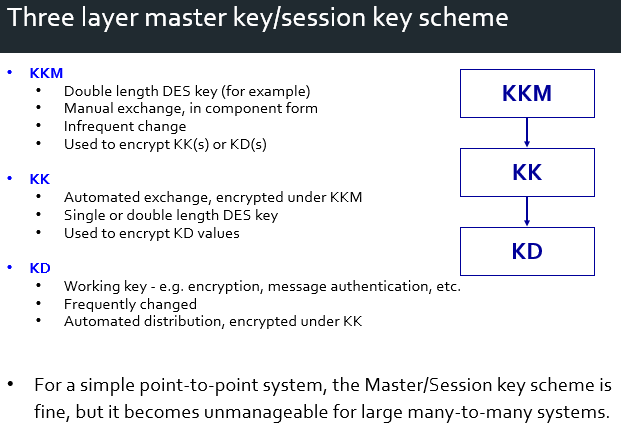
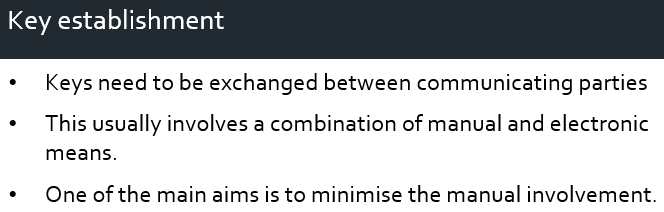


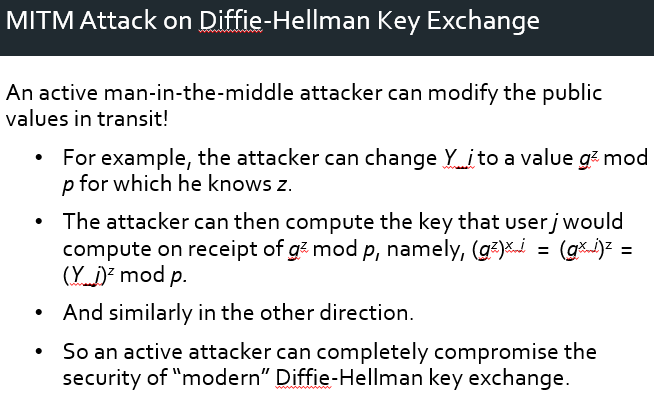
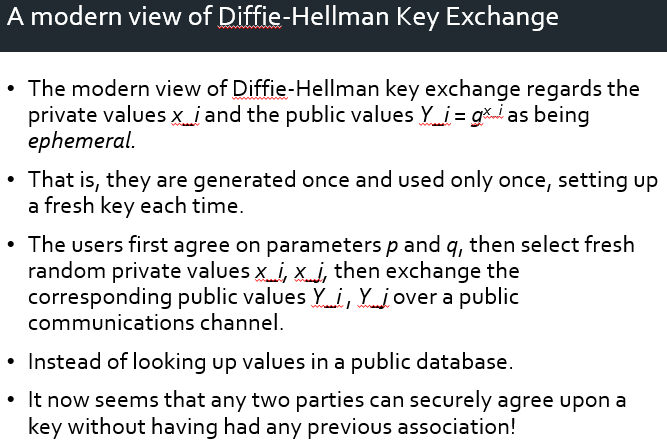
Key Generation

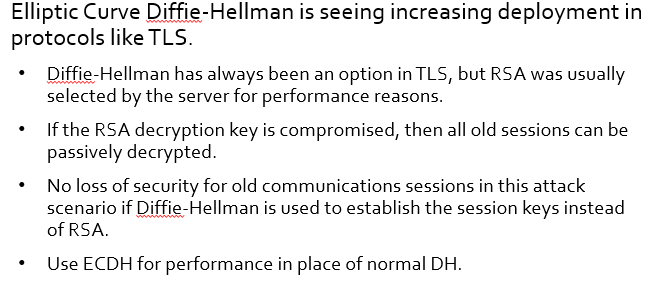
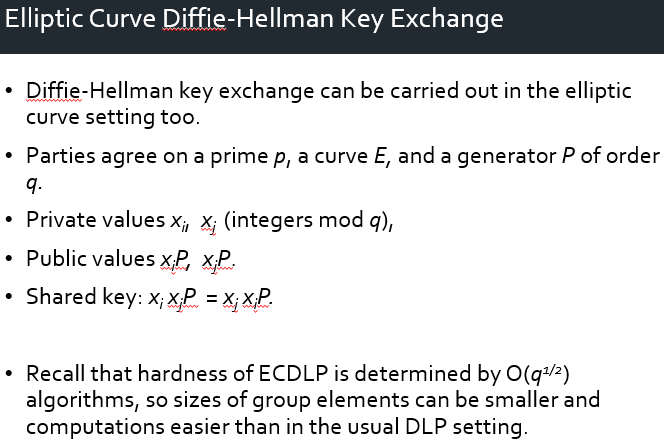
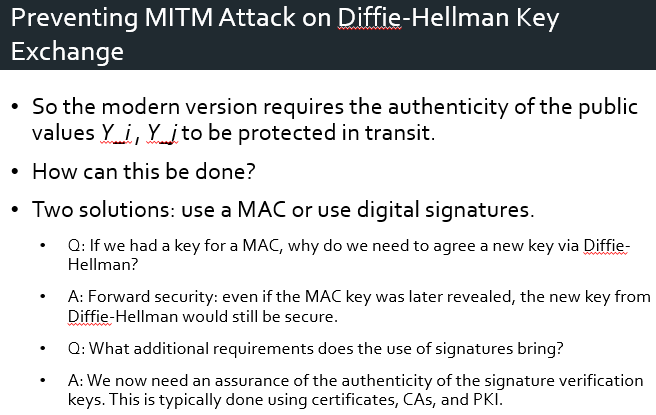






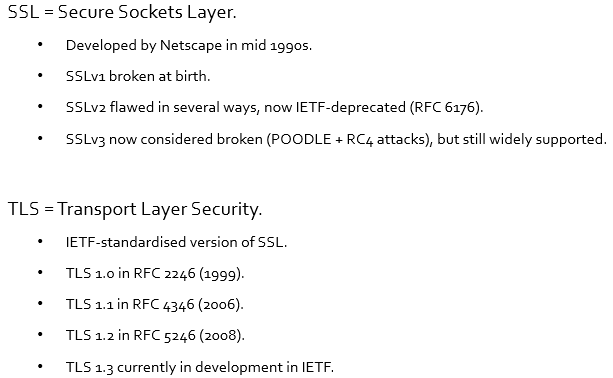


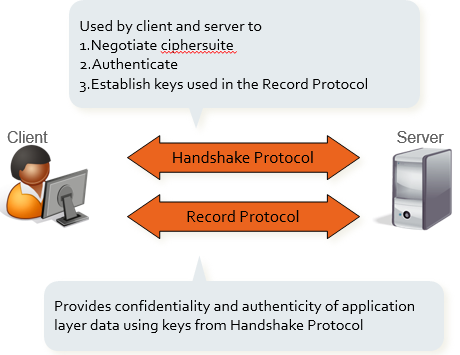




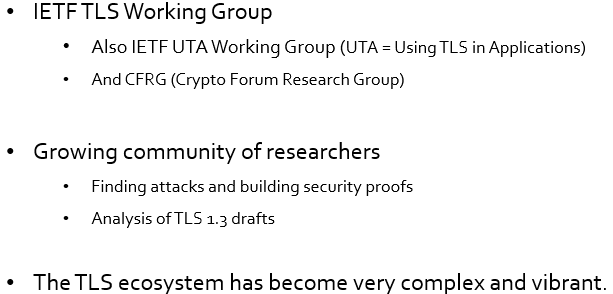
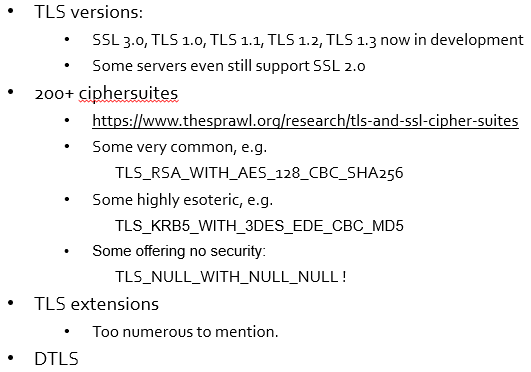
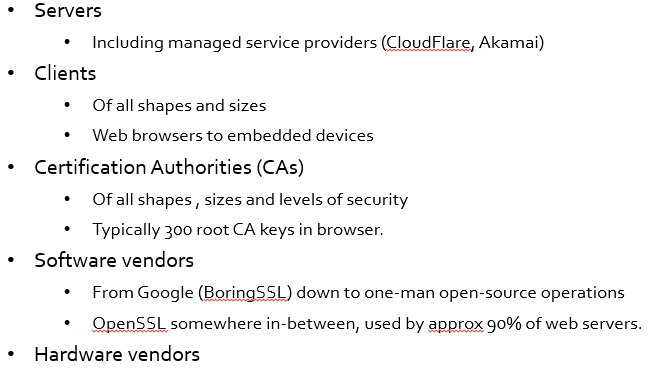
LECTURE 22 & 23 not tested.

**TLS – Transport layer security**

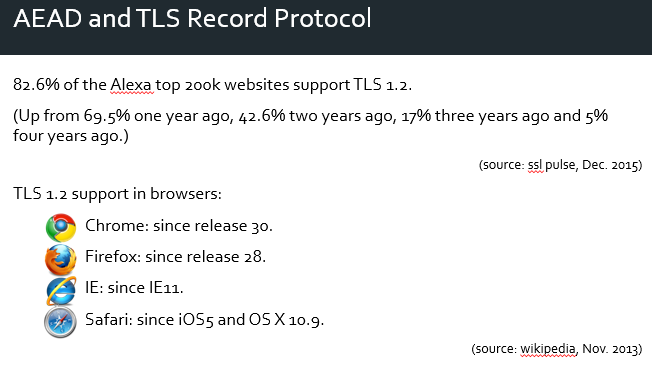
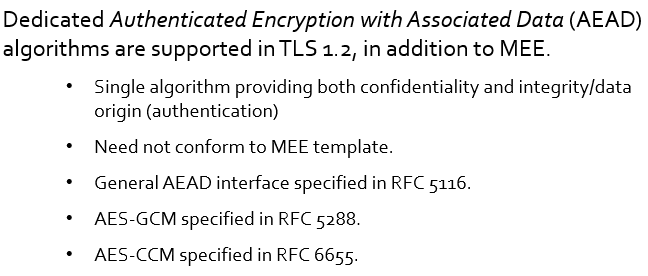
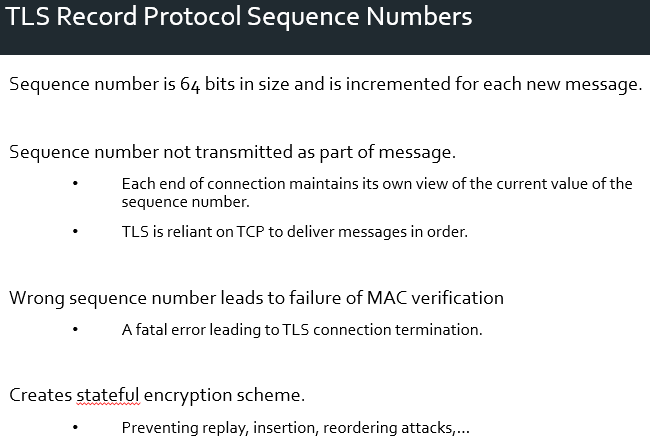
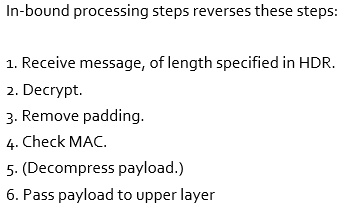
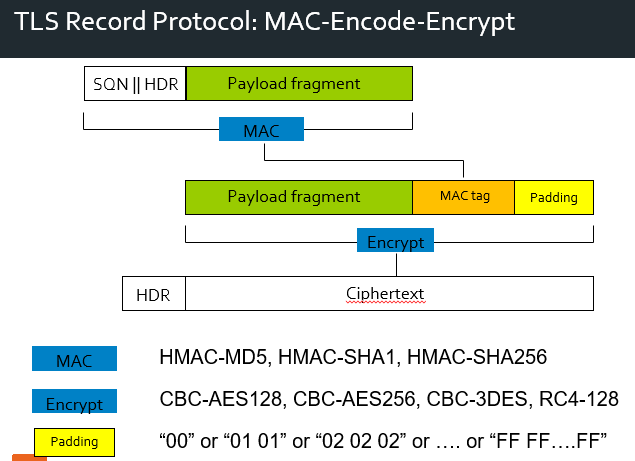
TLS, defacto, used everywhere



**TLS Ecosystem**



**TLS (Record Protocol) over TCP**



Increased usage for : ECDHE\_RSA\_AES\_128\_GCM\_SHA256

Attacks:

Beast

