PET

Definitions

dlp- discrete logarithm problem

Dsa- digitzal signature authentication

prf- pseudo random functions

prg - pseudo random generator

ppt- probabilistic polynomial time

dpt- deterministic polynomial time

tls- transport layer security

ssl-secure socket layer

kdf- key derivation function

aead- authenticated enc with associated data

cpa- chosen plaintext attack

tcp- Transmission Control Protocol

bgp- Border gateway protocol

hsts- HTTP strict transport security

rst- tcp reset

xss- Cross Site Scripting

fep- fully encrypted protocol

cdn- content delivery network

Internet and Privacy in Web

Network Topology

- Routers: direct Data between networks

- reduce need for direct connection between devices
- optimize data paths

- Servers:
- centralize resources and data storage
- handle requests from multiple devices
- enable shared services

- ISP: Internet Service Providers:
- Connect local networkds
- Provide Internet Access

- NSP: Network Service Providers:
- Internet Backbone: Connect ISPS
- Global coverage

How to send msg

- Redundant (multiple paths, no single point of failure)
- Dynamic routing (msg paths can change based on conditions →not always predicatble)
- Lacks privacy (intermediaries can see traffic)

IP

- Unique

- Every device on Internet has one
- Allows communciation

DNS

- Domain Name Resolution
- translates domain (ex.com) to IP
- Internets phonebook
- User types in domain→DNS searches for corresponding IP→Browser uses IP to communicate with Server
- Structure: Hierarchical root, Top-Level-Domain (.at) servers, authorative
- Plaintext Protocol, requests visible within same WIFI, to ISPs, in transit
- To encrypt: DNS over TLS, DNS queries over HTTPS
- problems still: recursive DNS Resolver, malicious DNS providers

HTTP

- HyperText Transfer Protocol
- Transfers data between web server and web client (browser)
- uses TCP/IP
- Clienst sends HTTP req (GET/POST) to server
- Server Respons with HTTP status code + content
- unencrypted, data visible for intermediaries
- entire page content, authentication tokens visible
- easy to monitor with transparent HTTP Proxie

- HTTPS: only leakes hostname in initial handshake

- deep packet inspection to monitor/censor
- even just trafic exposes metadate (destination servers, data volume)

- Attacks
- Rerouting: Data can take multiple paths (chooses best), BGP allows ISP to dynamically choose

Email Communication

- SMTP: Simple Email Transfer Protocl
- Send emails from clients to mail server

- IMAP: Internet Msg Access Protocol
- Clients can retrieve msg and manage mail directly on Server

- POP3: Post Officr Protocol3
- downloads mails from server to client, usually removing them from server

- How it works:
- Client sends SMTP req to server to send mail
- IMAP/POP3 used by client to retrieve incoming mail
- SMTP servers communicate with each other for deliveries
- Use TSL/SSL for secure transmission (does not ecnrypt msg for mail server, mail server can still access data)
- Encryption:
- ex PGP Pretty good privacy: public key for enc and signing

- ex S/MIME Secure/ Multipurpose Internet Email extension : Digital certificates for enc and signing

- Poor usability: complex, compatability, key management, users

- Confidentiality: protects mails from being read during transmission
- Integrity: Content not altered during transmission
- Authentication: Verifies identity of sender, preventing spoofing

Internet

- Global Network connected Computers
- supports data exchange
- mail, online games

Infrastructure

Web

- Service using the internet
- accesses websites through HTTP(S)

service on top of internet

Privacy in web

- ISPs can see: visited Domains (+ site content) by specific user
- Website can see: site content visited by specific user
- Ad networks / Social media see: Content on different websites visited by specific user

Fingerprinting & Censorship

Fingerprinting

- Collecting information about a devide, browser, connection to create unique id
- allows tracking/identification without cookies or IP

- can be based on:
- geolocation
- clock skew
- Battery
- Audio
- Screen resolution
- Installed fonts
- OS
- browser language

- Web browser can reveal this
- doesnt require consent

Canvas Fingerprinting

- tracking technique based on rendering images or text
- on HTML5 canva

- difference rendering (device, browser, GPU) creates fp (pixel data ex RGB values, extracted, varies because fonts, resolution, anti-aliasing (glättet Pixel smooth))
- Extracted pixel hash (we know hash, small change, completely different outcome)
- JS instructs browser to draw invisible images/text
- more difficult to block, because no data stored on device
- enables unconsented cross site tracking
- Countermeasures: privacy focused browsers /extension that blocks canvas access, TOR browser allerts when a website tries canvas fp

HSTS FP

- HTTP Strict Transport Security, forces browsers to always connect over HTTPS for secure connection
- websites send header for browser to instruct remember this for future visit
- Websites track users by checking if this setting has been cached for specific domains
- Different browsers store HSTS differently→ good for fp

- How fp:
- Website sends req to few test domains
- If browser forces HTTPS (due to caching) we have visited these domains before

- Tracking:
- Tracking agencies can create HSTS entries for subdomains, cached in browser to form consistent id (supercookie)

- Different combo subdomains → agencies create unique fp for user
- HSTS state remains even when cookies cleared
- any website can check cache by req res from subdomains

Censorship

- Goverment and ISPs control key internet infrastructure
- DNS manipulation and IP blocking
- Deep packet inspection to monitor and filter content

Detect what to censor

- destination IP addr
- server name indication
- DNS query content
- Proxy protocol signatures
- Filter for plaintext keywords

Block censorship

- IP null routing
- TCP Reset Injection
- DNS response inj
- HTTP redirect inj
- Packet dropping

TCP Connection Reset Attack

How TCP starts

1. Client SYN (synchronize package to initiate)

2. Server SYN-ACK (acknowledge your SYN)
3. Client ACK (acknowledges the acknowledgment, lets establish connection and complete handshake)
4. Data transission (gotta acknowledge the data as well, to ensure reliable delivery)
5. When done / issue TCP Reset (RST) packet send to determinate
6. immediately halts communication and frees system res so no further transmission

Attack on TCP

- Censor monitors traffic and injects a forged RST
- leads to termination
- Any data after is dropped, ends connection

Attacks on Network and Routing

- DOS: Overloading target server
- BGP Hijacking: Redirect traffic through compromised routes
- Network Disconnection: Cutting of entire regions of internet

Censhorship at Service Providers

- Search Engine Indexing, exclude certain website from results
- Deplatforming, remove accounts or users from platforms

- Cross site Scripting
- inject malicious script into web pages viewed by others
- used for stealing data/session cookies, but also for Censorship:
- modify or block access to certain content on targeted websites

- inject malicious JS →redirect users away from restricted content / modify page / replace content
- works at browser level, affects individuals not network level
- targeting specific users/ groups

URL Filtering

- implemented at ISP, black list to block
- in corporate networks / schools
- paired with TLS interception / DNS tampering to block encrypted websites
- Bypass: VPN, Tor, DNS over HTTPS

Chinese Firewall

- Blockes website and content (social media)
- Keyword filtering
- IP blocking
- Domain overblocking
- Traffic analysis and manipulation

- Targets:
- Foreign websites
- Politically sensitive
- Encrypted traffic

- How reinforced

- injects forged IPs to misslead or block users access to certain websites

- inconsistent
- not all forged injected equally
- some more frequently →inconsistent censorship

- Circumvention
- Identify forging patterns in IP
- detectable and bypassable with 99% success rate

Censorship Circumvention

- Network:
- Domain fronting: Route traffic through a different domain, makes it appear like accessing legitimate service, bypasses censorship
- Sending request to front domain that seems ok, while accessing a hidden target domain
- Both domains need to be hosted by same content delivery network
- masks true destination, difficult for censor to identify and block
- even if hidden target blacklisted, users can connect through front without interruption

1. Client requests allowed IP: gets IP of CDN or cloudservice
2. Client connects to CDN: establish connection using allowed domain in DNS and TLS handshake, is visible to other network observers
3. Within encrypted HTTPS req, clients specifies forbidden domain in HTTP host header (hidden from censorship we are in TLS)

- Cryptographic:
- Fully encrypted protocols (FEP): encrypt all data in transit, prevent interception and analysis by censors, ensure privacy and secure communication
- ensure all data is encrypted end-to-end
- Already encrypts at application layer, before it leaves users device
- Use obfuscation technique otherwise sus why completely encrypted
- Censor not sure→ allow or block everything? usually allow
- Censors rely on identifying patterns / keywords, get masked here

Cryptographic Preliminaries

Computationally Secure Encryption

Encryption

- function f(k,m)=c
- k key
- m message
- c cyphertext
- without k, not possible to decrypt c
- $f^{-}1(k, c) = m$ should be easy

Formal:

Definition (Private-Key Encryption Scheme)

A private-key encryption scheme $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is defined by a message space $\mathcal{M} = \{0,1\}^*$ along with the following three probabilistic polynomial time algorithms:

$k \leftarrow \mathsf{KGen}(1^\lambda)$: The key generation algorithm takes as input the security parameter 1^λ and outputs a key k chosen according to some distribution.

$c \leftarrow \mathsf{Enc}(k, m)$: The encryption algorithm takes as input the key k and a plaintext message $m \in \{0,1\}^*$ and outputs a ciphertext $c \in \mathcal{C}$. By $\mathsf{Enc}(k, m)$ we denote the encryption of the plaintext m using the key k.

$m = \mathsf{Dec}(k, c)$: The decryption algorithm takes as input a key k and a ciphertext c and outputs some plaintext m. We denote the decryption of the ciphertext c using the key k by $\mathsf{Dec}(k, c)$.

Perfect correctness requirement

Dec(k,Enc(k,m)=m

- Perfect unbreakable sec not possible due computational limits→balance robustness with efficiency
-

Security goal = desired outcome (Conf, Int, Authent)

Adversarial Model = assumptions about adversary's capabilities (access enc data, comp res)

CPA

chosen plaintext attack

Sec Goal: no extracting of meaningful info about plain from cyphertext

Adv Model: can choose plaintext, let it have enc, before and after actually attacking cyphertext → even if the attacker can get lots of examples how things are encr, they should still not know anything

Experiment CPA

$\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\Pi}(\lambda)$

$k \leftarrow \mathsf{Gen}(1^\lambda)$

$(m_0, m_1) \leftarrow \mathcal{A}^{\mathsf{Enc}(k,\cdot)}, \quad |m_0| = |m_1|$

$b \leftarrow \{0,1\}$

$c_b \leftarrow \mathsf{Enc}(k, m_b)$

$b' \leftarrow \mathcal{A}^{\mathsf{Enc}(k,\cdot)}(c_b)$

if $b' = b$ **return** 1

else return 0

1. Generate key
2. Adversary chooses 2 msg: m0, m1 with same length A ^ enc(k, .) means Adversary has access to Encryption Oracle
3. Challenger flips coin → 0 / 1
4. Based on flip m0 or m1 gets encrypted
5. Adversary only knows cyphertext, has to guess if m0 or m1
6. Guesses correctly 1
7. else 0

The probability that this game can be guessed correctly is ≤ 1/2 + negligible func

Pseudorandom functions (PRF)

Keyed deterministic func that behaves like truly random one as long as key unknown

Random input fixed size → larger quasi random output

essential for digital signatures, msg auth code

Distinguisher can test:

1. D gets blackbox they can query
2. Can be F(k, .) function for secret random key
3. or F(.) truly random func
4. Ist it PRF or truly random?

Let $Func_n = \{f | f : \{0,1\}^n \to \{0,1\}^n\}$ be the set of all functions mapping n-bit strings to n-bit strings.

Definition (Pseudorandom Functions)

Let $F : \{0,1\}^\lambda \times \{0,1\}^n \to \{0,1\}^n$ be an efficient, length-preserving, keyed function. F is a pseudorandom function (PRF) if for all PPT distinguishers D, there exists a negligible function negl such that

$$\left|\Pr\left[D^{F(k,\cdot)}(1^\lambda) = 1\right] - \Pr\left[D^{f(\cdot)}(1^\lambda) = 1\right]\right| \leq \mathsf{negl}(\lambda),$$

where the first probability is taken over uniform choice of $k \in \{0,1\}^\lambda$ and the randomness of D, and the second probability is taken over uniform choice of $f \in Func_n$ and the randomness of D.

F: two inputs, k and msg → produces cypher

Probability Distinguisher guessed correctly it was PRF - Probability it was truly random ≤ negligible func

Signature Schemes

Definition (Digital signature scheme)

A digital signature scheme Π_{DS} consists of three PPT algorithms $(\mathsf{KGen}_{DS}, \mathsf{Sign}, \mathsf{Vrfy})$ that are defined as follows:

$(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KGen}_{DS}(1^\lambda)$: The key generation algorithm outputs a pair of keys $(\mathsf{vk}, \mathsf{sk})$, where vk is the verification key and sk the corresponding signing key.

$\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, m)$: The sign algorithm on input a signing key sk and a message m outputs a signature σ.

$0/1 \leftarrow \mathsf{Vrfy}(\mathsf{vk}, m, \sigma)$: The verification algorithm on input a verification key vk, a message m, and a signature σ deterministically outputs a bit b, with $b = 1$ meaning valid and $b = 0$ meaning invalid.

KGenDS: output verification, signing key → I sign with signing, someone else can verify this was me with verification

Sign (sk, m): ouput signature s→ I sign a msg with sk and get an enc signature for m, like an official cryptographically protected stamp on msg

Vrfy (vk, m, s) checks if valid 1, else 0

correct if

Vrfy(vk, m, Sign(sk, m)) = 1

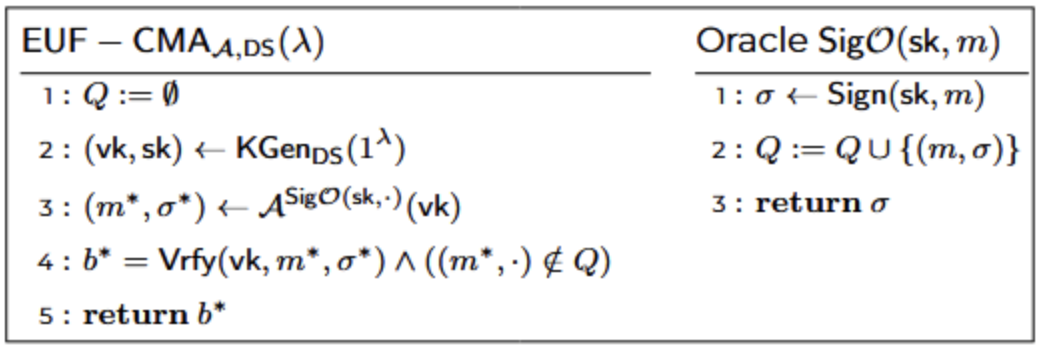
Existential Unforgeability under Chosen Message Attack

EUF-CMA

Adversary shouldnt be able to forge even one valid signature on any new msg

even if they are allowed to ask for signatures on chosen msg

hence we want them to forge a signature for a new msg



Adversary:

1. List of msg who got signed empty
2. Key gen (secret signing, public verification) Adversary only gets vk
3. Adv gets accessed to Oracle, and has vk → can ask for signature on any msg : Output msg m* signature

Meanwhile oracle:

1. If asked for signing, signs m with private signing and returns signature
2. Stores the pair (original msg, signature)in Q
3. returns signature

Adversary again:

4. if verifying (vk, msg, signature) is true and msg is not part of Q yet (meaning the oracle hasnt signed it) → true

Hash func

takes data of any size and transformes it into a fixed size unique fp

- efficient
- data integrity
- security

- used in
- Data storage and retrival
- digital signatures (ensures integrity, authenticity of documents and msgs)
- passwords (stored as hash)
- Blockchain, Cryptocurrencies
- File Integrity checks

Pigeon Hole Principle

Not collision resistant

Msg Input space >>> Hash Output space

butttt computationally infeasible to find

can be keyed (MACs, PRFs) s secret key, usually unkeyed

Input any lenght → Output fixed length

H^s (x) usually means s not secret

Test collision resistance

No efficient adv should be able to find 2 different inputs that produce same output

We define an experiment, where $\Pi = (\mathsf{Gen}, \mathsf{H})$ is a hash function.

$\mathsf{HashColl}_{\mathcal{A},\Pi}(\lambda)$

1 : $s \leftarrow \mathsf{Gen}(1^\lambda)$
2 : $(x_0, x_1) \leftarrow \mathcal{A}(s)$
3 : **if** $x_0 \neq x_1$ **and** $\mathsf{H}^s(x_0) = \mathsf{H}^s(x_1)$
4 : **return** 1
5 : **else return** 0

The adversary wins if $x_0 \neq x_1$ and $H^s(x_0) = H^s(x_1)$.

We write $\mathsf{HashColl}_{\mathcal{A},\Pi}(n) = 1$ if the output is 1 and we say that $\mathcal{A}$ succeeded.

1. Generate secret key
2. Adv gets key and tries to find collision
3. if not same msg, but hashes same
4. we win
5. we lose

Probability adv finds 2 such inputs ≤ negl

Schnorr Signature Scheme

based on discrete logarithm problem (DLP)

to securely authenticate digital msgs

efficient and simple

4 main alg

1. Setup
2. Key Gen
3. Signing
4. Verification

Schnorr Signature scheme

We have **four main algorithms**: Setup, Key Generation, Signing, and Verification.

$\textsf{Setup}(\lambda)$

1 : $(\mathbb{G}, q, g) \leftarrow \mathcal{G}(1^\lambda)$

2 : $\textsf{param} := (\mathbb{G}, q, g)$

3 : **return** param

Setup:

- **G**: a cyclic group where the discrete log problem is hard

Cyclic Group generated by generator g

- **q**: a large prime number (the group order)
- **g**: a generator of the group (a base point)

yes this g is generator

- all public no secret, because everyone needs to know them
- The group **G** contains elements like: g^0, g^1, g^2,... g^{q-1}

$\textsf{KGen}(\lambda)$

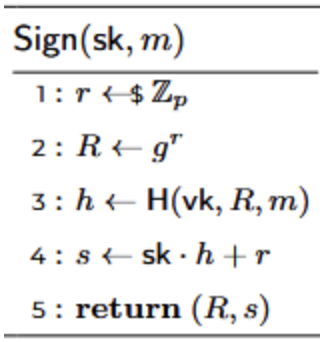
1 : $\textsf{sk} \leftarrow\$ \mathbb{Z}_q$

2 : $\textsf{vk} \leftarrow g^{\textsf{sk}}$

3 : **return** $(\textsf{sk}, \textsf{pk})$

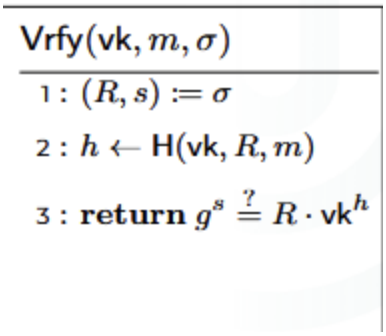
Key generation

- choose sk from 0,1...q-1
- set vk g^sk
- return key pair



Sign(sk, m)

- pick r from 0,1,...,q-1 (there is a typo I looked at wikipedia)
- Compute R (will be part of signature for randomness) R= g^r
- Now hash h= H(vk, R, m) = g^sk, g^r, m= g^(0-q-1), g^(0-q-1), m
- signature s= sk * *h+r= 0-q-1* * H(vk, R, m) + 0-q-1
- signature = secret (sk) * challenge (h) + r (randomness)
- Return (R, s)= (g^r , s)

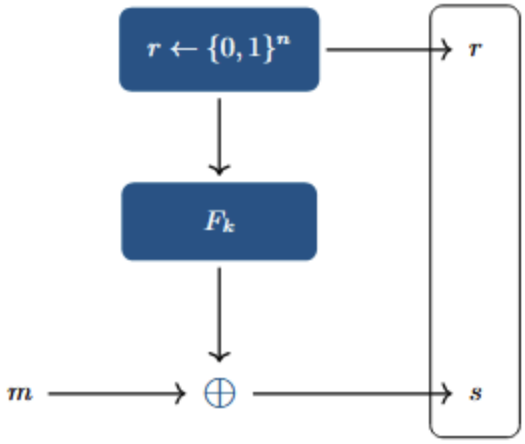


Verify (vk, m , Randomness + secret)

1. Parse signature
2. Recompute challenge (we know m, R, vk)
3. g^s= R*vk^h = g^sk*h+r =g^r *g^sk^h = g^(sk* h)* g^r= g^r * g^sk*h

Public Key Crypto and Security Reductions

CPA-secure enc from PRFs



1. We define a key k {0,1}^sec parameter

Enc:

2. Choose a r from same space
3. Compute c= (r,s) = (r, F(k,r) xor m) (s= F(k,r) xor m)

Dec:

4. Parse c as (r, s)
5. Compute m= F(k,r) xor s

If F is pseduorandom, then prev construction is CPA-secure private key enc scheme

Reduction proofs

Lets proof :

Let H be a known primitive. From that, we construct a new primitive G, which uses H in some way and has some specific properties

H is known to be hard to break

G is build on H (G uses H inside it, but is G secure?)

Trying

Breaking G → breaking H

but H secure

G must be secure aswell

We have adversaries

Say we have A, think of it as blackbox

A can break G

Based on A, we build B (B plugs A inside itself and internally simulates the world of G for A)

B can break H (If A can break G, I can use that to break H)

Because if A can break G, we can use A to build another attacker B, that breaks H

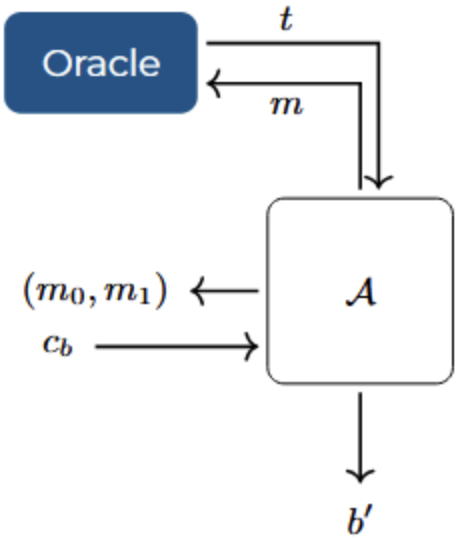
But H is secure, no one can break H (Contradiction

So B cant exist

→ A cant exist either

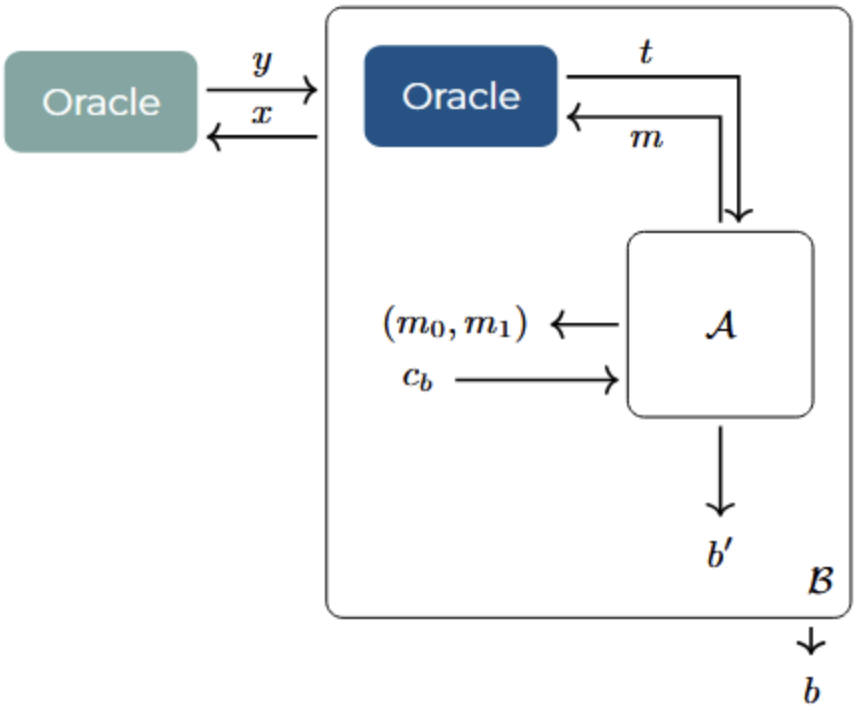
So G must be secure

As world:



- A produces m0, m1
- A can give oracle a msg to enc,
- gets t back (this is cyphertext)
- guesses which mo/ m1 produced cypher t
- b= 0/ 1

B simulates the Game for A



- B gets an instance of the H Challenge game and access to an oracle
- B simulates everything for A
- A wants to use Oracle, it sends it msg to what it thinks is the Oracle, but actually it is B, and B gives that input to its own oracle, and then answers A with the output of its own Oracle
- If A can break G → B can break H
- If such A exists → cant the secure primivite H would be broken
- G must be secure

Another Example

THEOREM

Let $F : \{0,1\}^\lambda \times \{0,1\}^n \to \{0,1\}^n$ a pseudorandom function. Then the function $F'(k,x) := F(k, x||0)$ is a pseudorandom function as well.

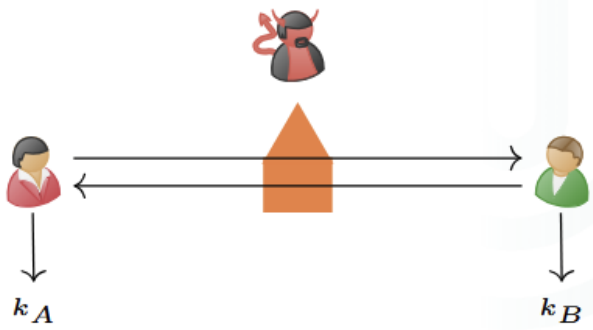
Key Exchange

Key exchange Eavesdropping Challenge

Tests if an adversary can tell the difference from session key to random key

$\mathsf{KE}^{eav}_{\mathcal{A},\Pi}(\lambda)$

1 : $(trans, k) \leftarrow \langle A(1^\lambda), B(1^\lambda)\rangle$
2 : $b \leftarrow \{0,1\}$
3 : **if** $b = 0 : k' \leftarrow k$
4 : **if** $b = 1 : k' \leftarrow\$ \{0,1\}^\lambda$
5 : $b' \leftarrow \mathcal{A}(k', trans)$
6 : **if** $b' = b$ **return** 1
7 : **else return** 0

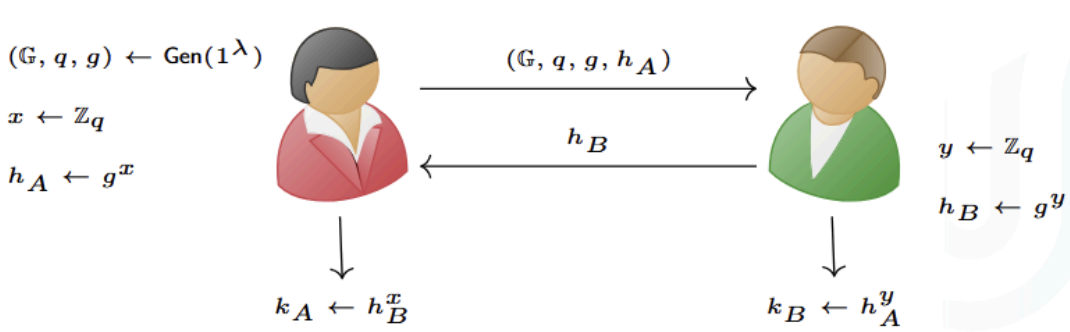


1. Adversary interacts with B running the exchange protocol, output: transcript (protocol of all msg) and real session key
2. Challenger flips coin
3. if 0 A receives real key
4. if 1 A receives fake key
5. A gets key (based on coinflip) and transcript, guesses where key is from
6. if correct we win
7. Loose

Key exchange protocol is secure in presence of Eavesdropper if

Probability Adv guesses if real key or fake key correctly ≤ 1/2 + negl

Diffie-Hellman Key Exchange



Correctness:

$$k_A = h_B^x = (g^y)^x = g^{x \cdot y}$$
$$k_B = h_A^y = (g^x)^y = g^{x \cdot y}$$

2 people want to communicate

Need a shared key, and a private key

Everyone can hear msgs but no one can compute the key

Setup Person A:

1. Cyclic Group G, q prime, g generater (g^0, g^1, ..., g^(q-1), all public info
2. pick x out of 0,1,..(q-1), this is their private key
3. public key: hA= g^x

Sends G, q, g, and public key (hA=g^x) to other person B

Setup Person B

1. picks y out of 0,1,...(q-1)
2. computes public key hB=(g^y)

Sends public key hB to A

Both computing shared key:

kA= hB^x

kB = hA^y

Why this works

kA= hB^x = g^y^x= g^yx

kB= hA^y = g^x^y= g^xy

So kA=kB

Eavesdropper sees: g, g^x, g^y but shared key g^xy

would need to compute x from g^x or y from g^y

→Discrete Logartihm problem is hard

Diffie Hellman is not encryption, but shared key can be used for symmetric cryptography

Security assumptions:

Remember our eavesdropping challenge, Adv has to guess real key or randomly generated

for this lets say not randomly generated but group element (from G)

Can they distinguish it?

No it is hard relative to the key exchange seen before, we know the other one is secure → this one aswell

Definition (Decisional Diffie-Hellman)

The Decisional Diffie-Hellman (DDH) problem is hard relative to Gen if for all PPT algorithms $\mathcal{A}$ there exists a negligible function negl such that

$$\Pr[\mathcal{A}(\mathbb{G}, q, g, g^x, g^y, g^z) = 1] - \Pr[\mathcal{A}(\mathbb{G}, q, g, g^x, g^y, g^{xy}) = 1] \leq \mathsf{negl}(\lambda),$$

where in each case the probabilities are taken over the experiment in which the group generation algorithm $\mathsf{Gen}(1^\lambda)$ outputs $(\mathbb{G}, q, g)$, and then uniform $x, y, z \in \mathbb{Z}_q$ are chosen.

Discrete Log Assumption

Gen is a group generation Alg

1. We generate cyclic Group G, prime q, generator g
2. Pick random h from Group G
3. Adversary gets Group, prime, generator, h we randomly picked, A tries to guess x such that g^x= h
4. if guessed win else loose

probability of guessing such an x is ≤ negl

Why is it that safe?

Cyclic groups of prime order:

- DLog is the hardest in such groups (g^0, g^1, .. g^(q-1))
- we have exactly q elements
- Pohlig-Hellman algorithm can solve it for small prime factors (ex 30=2*3*5), q is prime, so we have no factors at all → problem is hard

- finding a generator g is easy, every element is

(Subgroups of) Zp*

- Multiplicative group % p
- group order p-1, not necessarily prime
- DLog and DDH may be easier in Zp* directly if p-1 has small factors
- Therefore use subgroups:
- Pick p and q such that p=r*q+1 (p,q prime, r int)
- Then Zp* has subgroup of order q
- Working in this prime order subgroup hard for DLog and DDH

Why does this matter?

- Working in Zp* directly and p-1 has small prime factors (make it easy to break)
- By working in subgroup of prime order q, we avoid these attacks

Public Key Enc

Definition (Public-Key Encryption)

A public-key encryption scheme consists of three PPT algorithms:

$(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KGen}(1^\lambda)$: The key generation algorithm outputs a pair $(\mathsf{sk}, \mathsf{pk})$. We refer to the first of these as the private key and the second as the public key.

$c \leftarrow \mathsf{Enc}(\mathsf{pk}, m)$: The encryption algorithm takes as input a public-key pk and a plaintext message m and outputs a ciphertext $c \in C$.

$m \leftarrow \mathsf{Dec}(\mathsf{sk}, c)$: The decryption algorithm takes as input a private-key sk and a ciphertext c and outputs some plaintext m. We assume w.l.o.g. that Dec is deterministic.

sk private (think of secret key), used for decryption

pk public, used for encrytion

Correct if

Dec(sk, Enc(pk, m))= m

Experiment

Public Key Enc Scheme

$\mathsf{PubK}^{\mathsf{cpa}}_{\mathcal{A},\Pi}(\lambda)$

1 : $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Gen}(1^\lambda)$

2 : $(m_0, m_1) \leftarrow \mathcal{A}(\mathsf{pk}), \quad |m_0| = |m_1|$

3 : $b \leftarrow\$ \{0,1\}$

4 : $c_b \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b)$

5 : $b' \leftarrow \mathcal{A}(c_b)$

6 : **return** $b' \stackrel{?}{=} b$

1. Generate key pair
2. Adv gets public key and can enc m0 or m1 (both same length)
3. Challenger flips coin
4. Enc with public key m0 or m1 based on flip
5. Adv has to guess which msg was enc
6. 1 if guessed correct otherwise loose

El Gamal Enc

KGen(λ)	Enc(pk, m)	Dec(sk, c)
1 : $(\mathbb{G}, q, g) \leftarrow \mathsf{Pgen}(1^\lambda)$	1 : $(m \in \mathbb{G})$	1 : $\langle c_1, c_2 \rangle := c$
2 : $x \leftarrow\$ \mathbb{Z}_q$	2 : $y \leftarrow\$ \mathbb{Z}_q$	2 : $m := \frac{c_2}{c_1^x}$
3 : $h := [x]$	3 : $c_1 := [y]$	3 : **return** m
4 : $\mathsf{sk} := \langle \mathbb{G}, q, g, x \rangle$	4 : $c_2 := \mathsf{pk}^y \cdot m$	
5 : $\mathsf{pk} := \langle \mathbb{G}, q, g, h \rangle$	5 : **return** $\langle c_1, c_2 \rangle$	
6 : **return** (sk, pk)		

Key Gen

1. Choose group G, q prime, g generator
2. pick sk x from 0,1,..q-1
3. h = g^x

4. sk is defined under Group g, q prime, g generator, x randomly chosen (actual key)
5. same for private key h = g^x
6. return key pair

Enc (pk, m)

1. Choose m from group G
2. pick a private key y from 0,1,.., q-1
3. compute c1= g^y (masking value
4. compute c2= pk^y * m = h^y *m =(g^x)^y * m = g^(xy) * m (msg hidden under g^xy
5. return (c1, c2)

Dec (sk, c)

1. recover c1, c2 from c
2. compute m, m = c2/c1^x, where x = sk, works because c1^x= (g^y)^x = g^xy, c2= pk^y *m =h^y *m = g^xy * m, so we compute (g^xy *m) / g^xy =m
3. return m

El Gamal is probabilistic

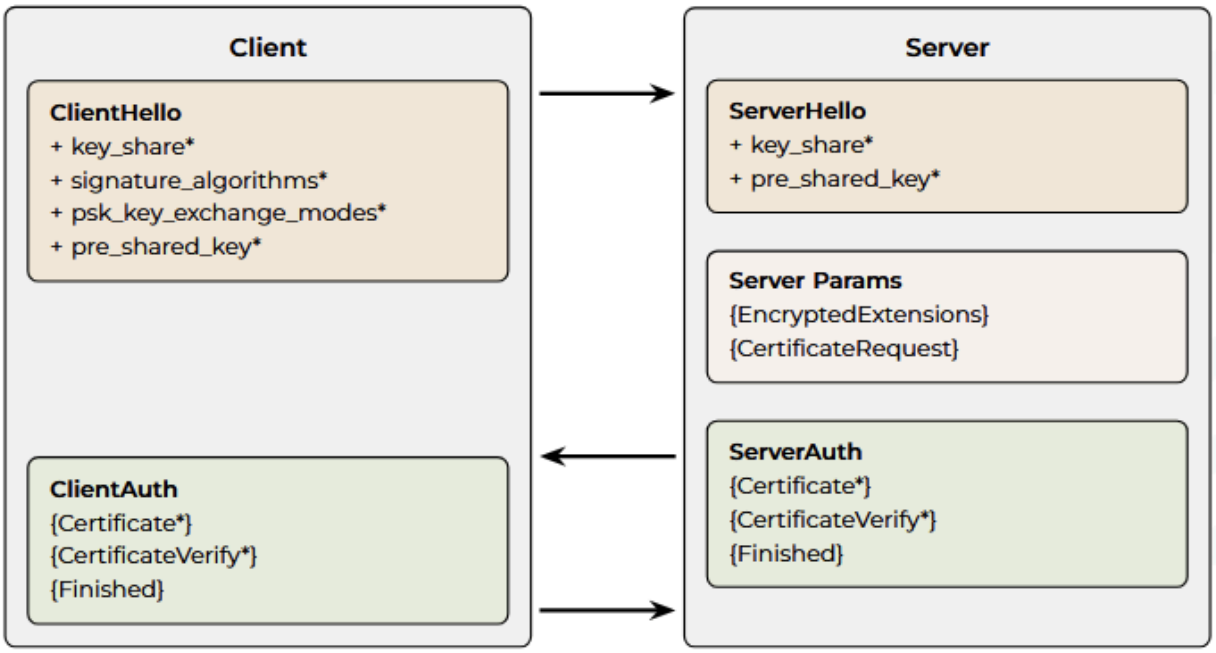
Enc same msg twice → different cypher because of randomly choosing y freshly each round

TLS

- Transport Layer Security
- protocol for secure communication over computer network
- encrypts data for
- confidentiality (unauthorized parties cant eavesdrop during transmission)
- integrity (data not tampered with during transit)
- authentication (verifies identity of both communicating partners, no man in the middle or impersonation)

Handshake

- Authentication (mutual, optional)
- Negotiate: Cryptographic modes (algo) + param
- establish shared keying material (shared secret for secure communic)



1. Client Hello

- key share (transient keys client would use (for ex. DDH , used to derive session key))
- signature algos (supported algos fpr verifying certifcates=
- psk key exchange modes (modes supported for pre shared key)
- pre shared key (identity of prev shared key, ex to resume session)

Lets server know which protocols, algos and keying option client has

provides client part of key exchange if ephemeral (transient, meaning not forever) keys used

2. Server Hello

- key share (servers ephemeral key for key exchange, combine with client key to compute session key)
- pre shared key (if resume session)

At this point both have shared a key g^x and g^y

so both can compute shared secret g^xy

3. Server Params

- Enc extensions (additional param / capabilities, enc with negotiated key)
- Certif req (asks client to authenticate themself)

4. Server Authenticate

- Certificate (identifies server)

- Certificate Verify (proof server owns private key corresponding to its certificate)
- Finished (MAC (Msg Authentication Code) ober handshake msgs to confirm integrity)

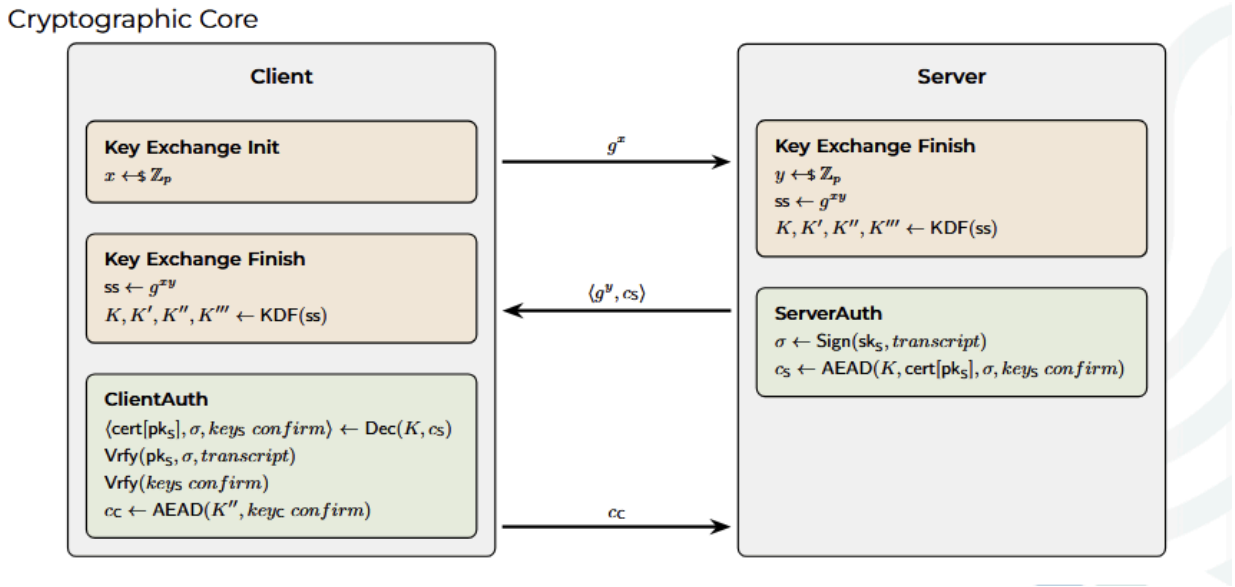
5. Client Authenticate

- Certif (Clients Certif)
- Certif Verifiy (proof client owns its private key)
- Finished (MAC)

Completed Handshake

Both sides now share session key and can start encr communic

Cryptographic core more detailed



1. *Client* picks random private key x, sends g^x to server

1. *Server* picks random private key y
2. computes shared secret g^xy
3. shared secret used for key derivation function to generate multiple keys
4. Client gets x^y and can compute that aswell

1. *Server* authenticate, it signs handshake transcript with its signing key (private not part of that exchange, proves identity)
2. encrypts following package = (K (session key derived from DDH with ss = g^xy), cert[pks] (server certificate of public key), **σ** servers signature over transcript, keysS confirm (servers key confirmation (proves it derived same session key as client), gets encrypted with AEAD (authenticated enc with associated data) → only client with same derived key can encrypt
3. sends g^y and cs to Client

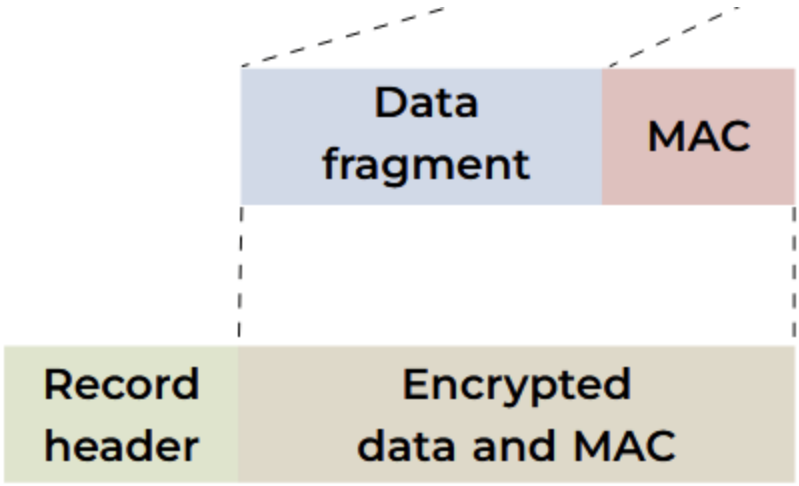
σ = "I, the server, own this certificate and I endorse all handshake messages so far."
cS = "I send this signed statement and my key confirmation encrypted under the shared DH key so only the client can read it."

1. *Client* authenticate, for mutual TLS authentication
2. with help of K (Key derived from function from shared secret) Dec cs (much data that server enc), gets certif of public key of server, **σ** servers sig over transscript, keys confirm (same key computed
3. Verifies with public key of server, that signature over transcript is valid → meaning server owns its private key (signing key for signatures)
4. Verifies they have computed the same shared secret g^xy
5. Client sends own confirmation: keyC confirmed enc with K'' (derived from func) and encr with AEAD

Record

- uses established params
- protects traffic between peers
- divides traffic into series of **records,** independently protected using traffic keys
- ensures confidentiality, integrity
- applies enc and authent to transmitted rec

Structure of Record:



We get data

→ gets split

each data fragment (2^14 bytes) gets Msg Auth Code (includes sequence nr so we get can restruct original data)

→ enc data and mac

Record header: content type, version, length

TSL in practice

Abstract Primitive	Construction	Based on	Security Guarantee
AEAD	AES, -GCM, -CCM; ChaCha20-Poly1305	Blockcipher; Mode of Operation; Streamcipher	Message-Indistinguishability, Integrity
Signature Algorithm	RSA, -PKCS, -RSS; EdDSA		Authentication, Integrity
Key Derivation Function	HKDF (protocol based on hash functions)	HMAC (Hash function)	Key Indistinguishability

HTTPS

- TLS secures web traffic
- enc communic between browsers and web servers
- ex Access bank website: Browser connects to bank ws, TLS handshake → servers identitiy verified, following communic enc

Common attacks:

- Man in the Middle: Attacker intercepts and alters communic
- Downgrade: Forcing connection to use older TLs vers (less secure)
- Heartbleed: Vuln in OpenSSL lead to library allowing data leak, reading up to 64KB of data of connected server or client was possible (leaked private keys, passwords, session tokens), Heartbleed because TLS used to send sth like are you here? send me 5 these 5 letters “alive”, attackers did alive? send me 40004 letters “alive” response was alive_more_data_here, **payload** not length validated, now regenerating private keys and reiusse SSL certif

- TLS Stripping: HTTPS→ HTTP

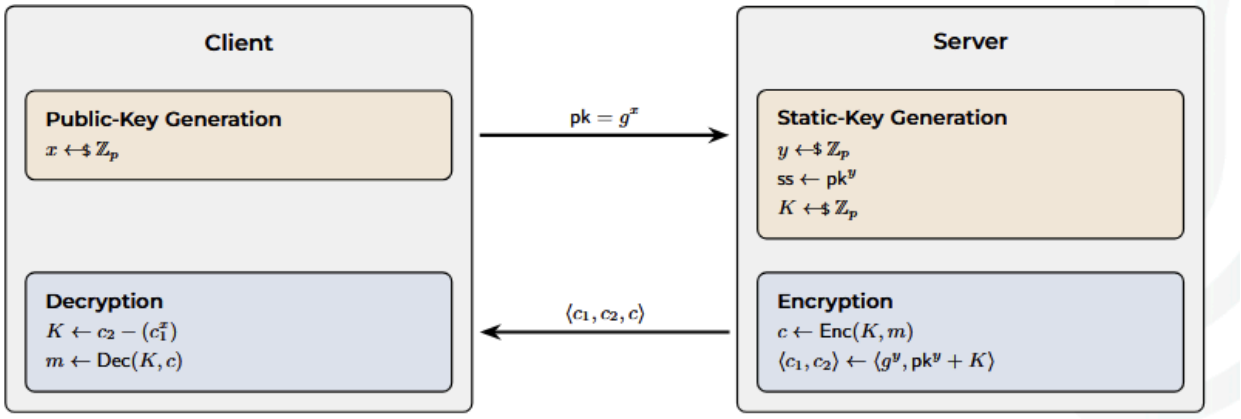
How to stay safe:

- Use modern versions
- Certificate pinning (ensure known certif is used by server)
- HSTS (strict transport sec) enfore HTTPS
- Keep libraries updated

Proofing TLS secure

complicated. why?

- protocol complex (handshake, record protocol, key deriv), all these tightly intertwined, modelling formally difficult
- real world uses different impl, unforseen vulnerabilities
- many sec goals (authent, confid, forward secrecy, resistance to tampering), proving all this single framework complex
- backwards compatability for legacy features, complex
- proofs rely on multiple cryptographic assumpt, holding true under all conditions? challenging



1. Client generates public key (picks x and sends server g^x)
2. Server picks y and creates shared secret ss g^xy = pk^y
3. Server picks random K which gets used for sym enc
4. Server sym enc m with K, produces c1 (g^y) and c2 (g^xy + K) sends to client
5. Client computes K by c2-c1^x and can dec c with K and gets m

More detail:

$\mathsf{HybridKGen}(\lambda)$	$\mathsf{HybridEncrypt}(\mathsf{pk}, m)$	$\mathsf{HybridDecrypt}(\mathsf{sk}, \langle c_1, c_2, c \rangle)$
$1: \mathsf{sk} \leftarrow\$ \mathbb{Z}_p$	$1: K \leftarrow\$ \mathbb{Z}_p$	$1: K \leftarrow c_2 - c_1^{\mathsf{sk}}$
$2: \mathsf{pk} \leftarrow g^{\mathsf{sk}}$	$2: c \leftarrow \mathsf{AESEncrypt}(K, m)$	$2: m \leftarrow \mathsf{AESDecrypt}(K, c)$
$3:$ **return** $(\mathsf{sk}, \mathsf{pk})$	$3: y \leftarrow\$ \mathbb{Z}_p$	$3:$ **return** m
	$4: \langle c_1, c_2 \rangle \leftarrow \langle g^y, \mathsf{pk}^y + K \rangle$	
	$5:$ **return** $\langle c_1, c_2, c \rangle$	

HybridKGEn

1. pick secret key

2. public k =g^sk= g^x
3. returns key pair

HybridEnc(pk, m)

1. Pick K
2. Use K to enc m wirh AESEnc → get c
3. Pick y
4. c1=g^y, c2= g^xy + K
5. return c1, c2, c

HybridDec(sk, c1, c2, c)

1. Compute K = c2- c1^sk = pk^y + K - g^y^sk = g^xy + K - g^xy
2. Now we have K , compute m by AESDec(K, c)
3. return m

TLS does hybrid enc

uses asym public key enc for shared secret (Diffie Hellman)

uses sym enc (AES) to enc actual msg m with K

How entagled

Cyphertexts c1, c2 and

c1 (g^y) and c2 (g^xy + K) encapsulate K and g^x (public key)

So K (used for sym Enc) is msg of El Gamal Ciphertext

When we want to reduce to CPA, we do not know K

Proof TLS secure

Outline:

2 steps

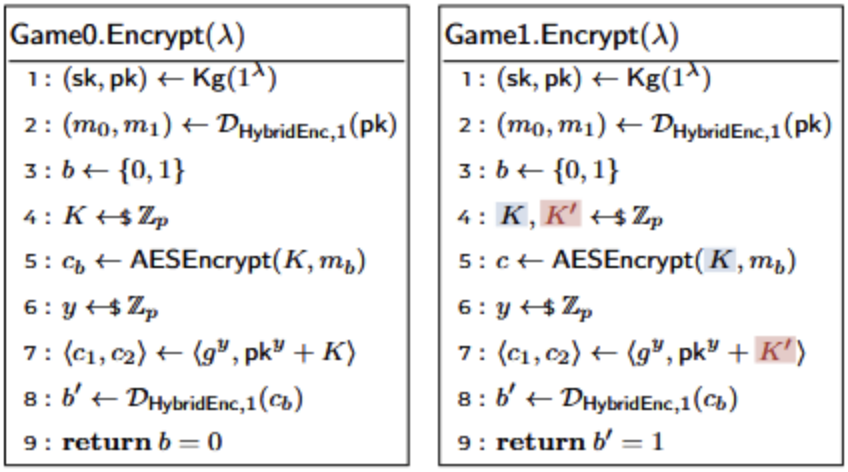
1. Modify ⟨c1, c2, c⟩ = ⟨g^y , pk^y (g^xy) + K′ , AESEncrypt(K′ , m)⟩, difference: K′ instead of K
- Seperates ciphertexts since K no longer part of encapsulatiobn
- This modif makes dec impossible
- This gap only discoverable by Adv breaking CPA sec of El Gamal
- We show, using reduction from gap between Cyphertext to CPA sec of El Gamal, that difference between original cipher and this cipher is negligible (indifferent)

2. Reduce from modified cyphertext to CPA sec of AES
- Use enc oracle
- ⟨c1, c2, c⟩ = ⟨gy , pky · K′, EncO(m)⟩
- reduction does not know effective enc key

These 2 steps should show that El Gamals CPA sec and AES CPA sec bounds to probability of breaking CPA sec of hybrid enc

We assume El Gamal and AES to be secure → Hybrid secure aswell

1st reduction, encopling K with K′



Game0 real game with AES K same as in enc inside c1, c2

1. Generates El Gamal keypair
2. Adv chooses 2 msgs to encr with public key
3. Flip coin
4. Choose K
5. Based on flib enc m0 or m1 with chosen K
6. Choose y
7. set c1 = g^y, c2= pk^y (g^xy) + K) (El Gamal cyphertext and AES cipher same key)
8. Give adv cypher, has to guess
9. return b=0

Game1

1. Generates El Gamal keypair
2. Adv chooses 2 msgs to encr with public key
3. Flip coin

4. Choose K and K′
5. based on flip enc mb with **K**
6. choose y
7. Set up c1, c2 as before but with **K′**
8. Give adv cypher, has to guess
9. return b′ =1

Noticed our returns weird?

If Adv cannot tell game0 from 1→ El Gamal Ciphertext reveals nothing about K

In game0 b=0 sometimes true sometimes false, nothing to do with actual gzess

In game1 b′=1 also sometimes true false and does not depend on how good guess

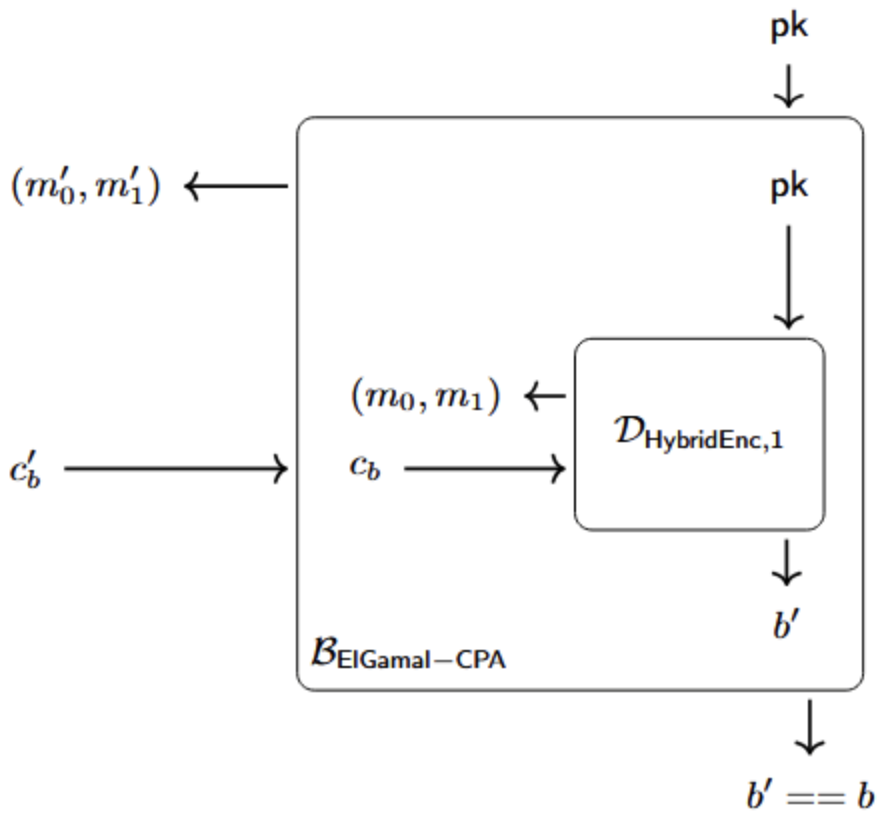
why?

Because goal is to compare

Prob (game0 retunrs 1) - Prob(game1 returns1)

Pr[Game0 returns 1] = 1/2
Pr[Game1 returns 1] = Pr[b′ = 1]

| Pr[Game0 = 1] – Pr[Game1 = 1] |= | 1/2 – Pr[b′ = 1] |



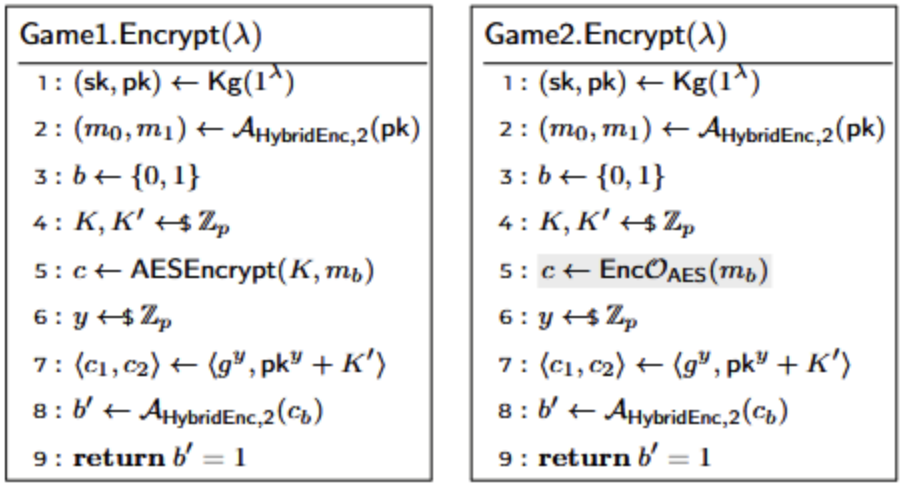
Adv gets

- public key, produced cyphertexts (doesnt know which is which)

Adv can

- create msg, ask for enc, guess which cyphertext got enc

2nd reduction, using Oracle instead of Enc ourselves



Game1

Same as before

Game2

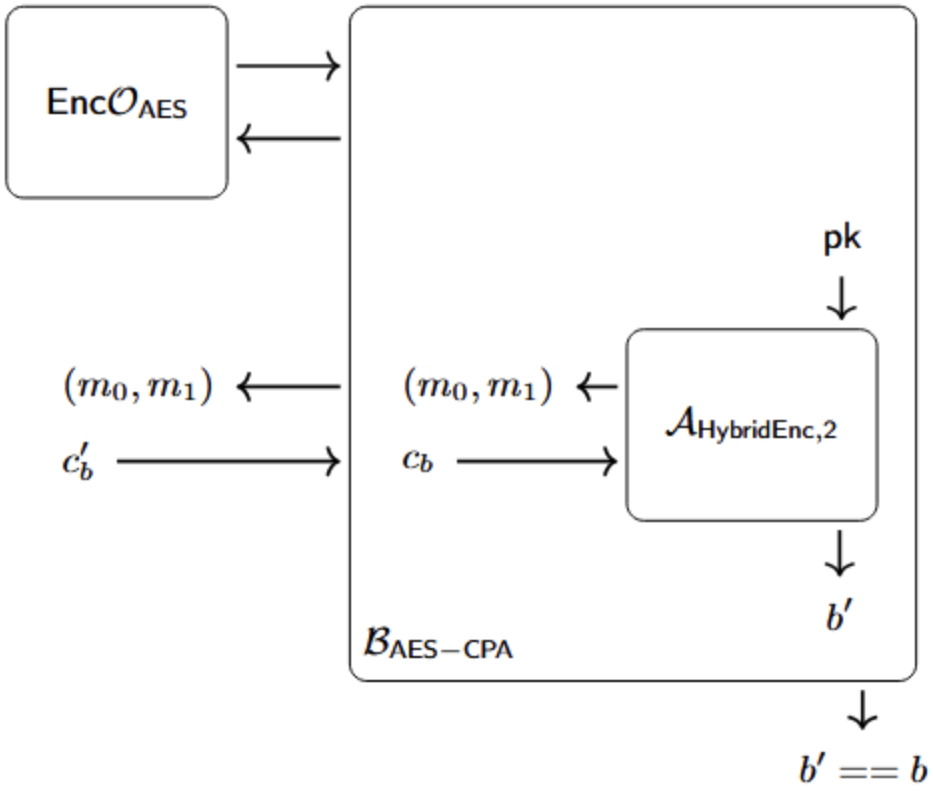
also same

except

5. Enc with Oracle

Can adversary see difference between real enc and oracle?

return tests if Adv behaved the same in both games



AES on its own is not CPA-secure, achievable only when used in an appropriate mode of operation, such as CBC with random IVs, CTR mode, or authenticated encryption modes like GCM