Final Review: (Almost) Everything you need to know to not fail the final exam

Systems Security Lab

Security Design Principles

From Saltzer and Schroeder's 1975 (!!) article "*The Protection of Information in Computer Systems*"

1. Least Privilege
2. Fail-Safe Defaults
3. Economy of Mechanism
4. Complete Mediation
5. Open Design
6. Separation Privilege
7. Least Common Mechanism
8. Psychological Acceptability
9. Defense in Depth

Interestingly, most of these principles still hold as you will see

Access Control Lists (ACLs)

- ACL: Store access control matrix by column
- Explains *WHO* can access Insurance data

	OS	Accounting program	Accounting data	Insurance data	Payroll data
Bob	rx	rx	r	—	—
Alice	rx	rx	r	rw	rw
Sam	rwx	rwx	r	rw	rw
Accounting program	rx	rx	rw	rw	rw

Systems Security Lab

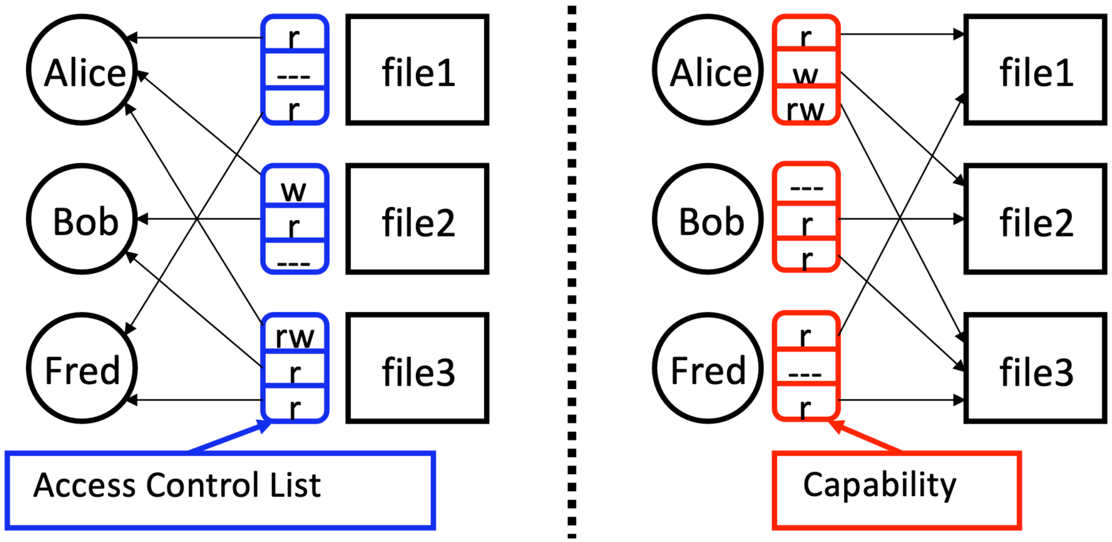
Capabilities

- Store access control matrix by row
- Defines *WHAT* Alice can access

	OS	Accounting program	Accounting data	Insurance data	Payroll data
Bob	rx	rx	r	—	—
Alice	rx	rx	r	rw	rw
Sam	rwx	rwx	r	rw	rw
Accounting program	rx	rx	rw	rw	rw

ACLs vs Capabilities

- Note that the arrows point in opposite directions
- With ACLs, we need *file-user association*



Confused Deputy

- Two resources
  - Compiler and **BILL** file (billing info)
- Compiler can write file **BILL**
- Alice can invoke compiler with a debug filename
- Alice not allowed to write to **BILL**

❑ Access control matrix

	Compiler	BILL
Alice	x	—
Compiler	rx	rw

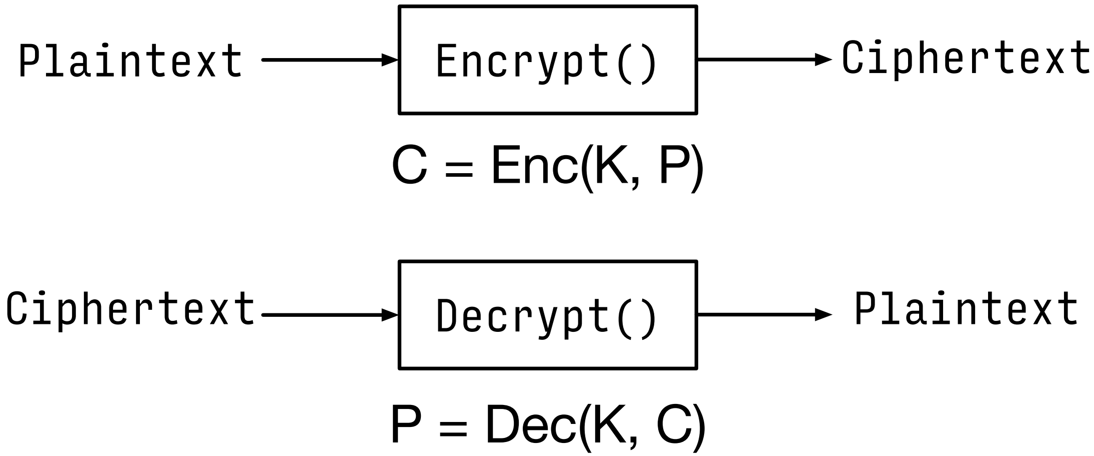
Crypto

How to Speak Crypto

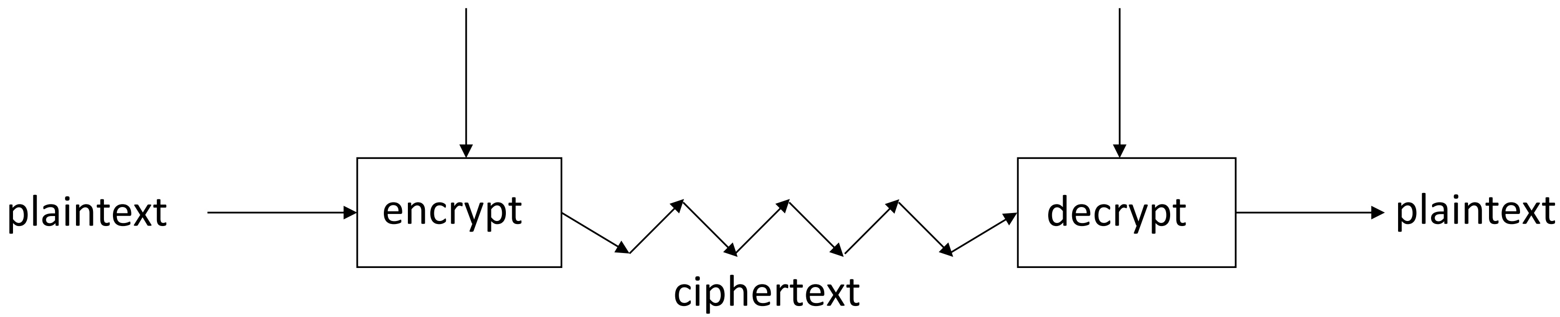
- A *cipher* or *cryptosystem* is used to *encrypt* the *plaintext*
- The result of encryption is *ciphertext*
- We *decrypt* ciphertext to recover plaintext
- A *key* is used to configure a cryptosystem
- A *symmetric key* cryptosystem uses the same key to encrypt as to decrypt
- A *public key* cryptosystem uses a *public key* to encrypt and a *private key* to decrypt

Symmetric-Key Cryptography

- Uses the same key for encryption/decryption

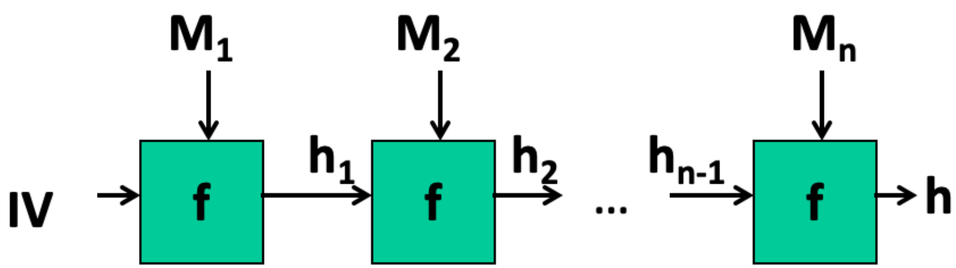


- Assumption: Sender and Receiver already have a shared secret key



Hash Functions

- Hashing is a one-way only encryption
  - No such thing as unhashing or dehashing
- There is <u>no key</u> used in hashing
  - $H(m) == h$ vs. $Enc(key_{enc}, m) = c$
- Fast computation time



Hash Functions

- Purpose: produce a fixed-size "fingerprint" or digest of arbitrarily long input data
- Hash passwords such that password plaintext need not be saved on the service or server
- <u>To guarantee integrity</u>

MAC

- Message Authentication Code (MAC)
- One-way Function (Basically a Hash function with a key) that creates a message *digest*
  - e,g, MAC(k,m) = d
- A digest is appended at the end of the message, so that the receiver can verify it

MAC vs Hash

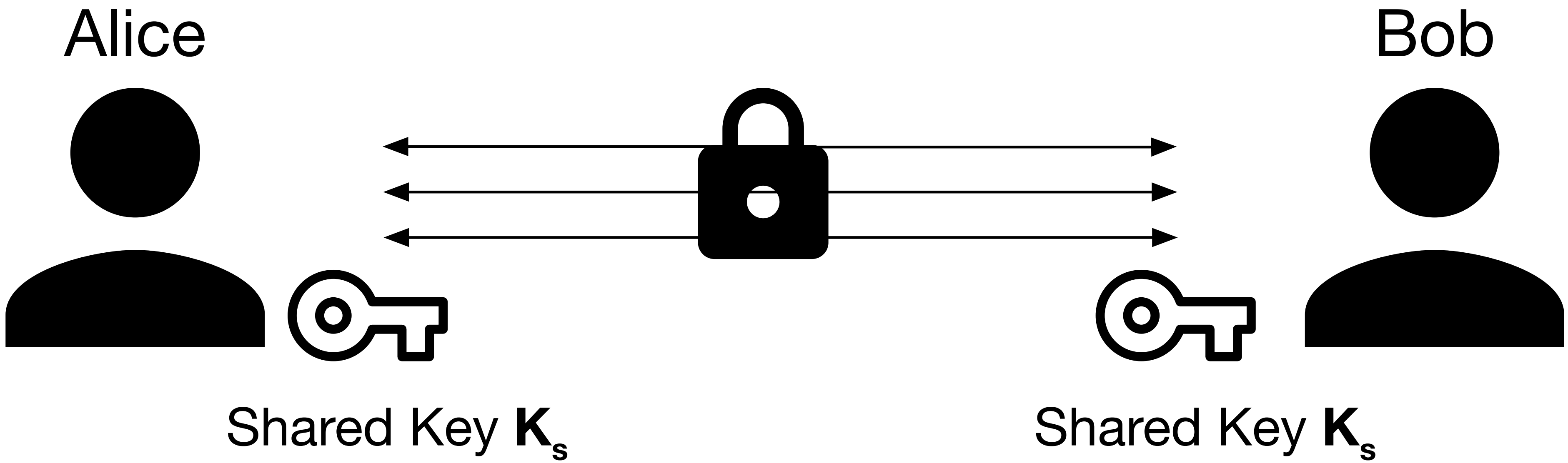
- Key is used during computation
- Ensures integrity and authenticity of the message
- A shared key is need to verify a MAC

- Key is not used during computation
- Ensures only integrity
- Everyone can verify a hash

History of Public-Key Cryptosystems

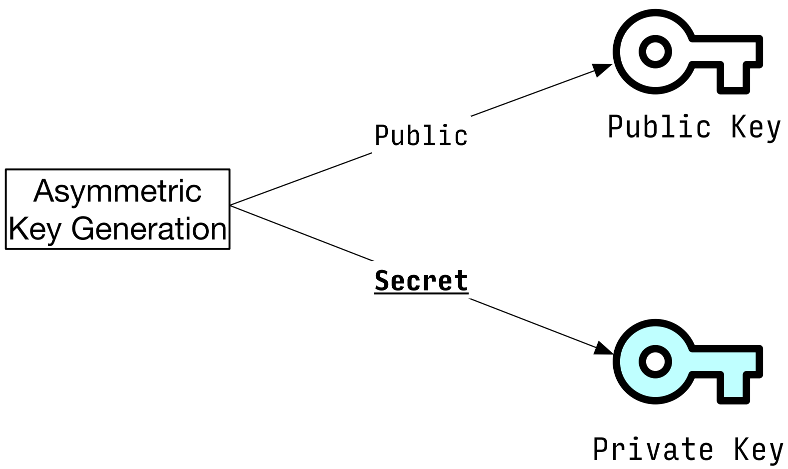
- Before the mid 1970s all cipher systems were symmetric key algorithms.
- Symmetric keys are still widely used today
- known to 2-3 magnitudes faster than asymmetric (a.k.a public-key) algorithms.
- Why was public-key cryptosystems were such a breakthrough?

The Key Exchange Problem



- What is the problem here?
- Look at the title of the slide ☺

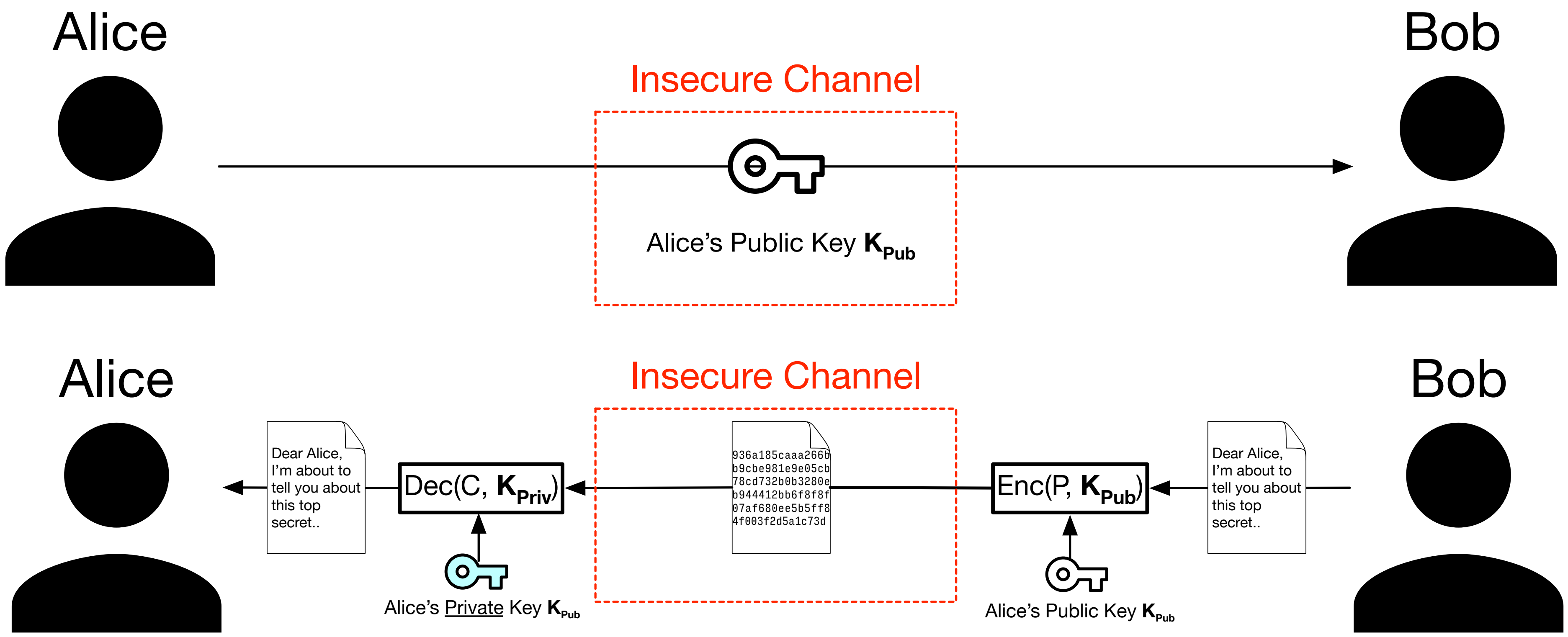
Public-Key (Asymmetric) Cryptosystem



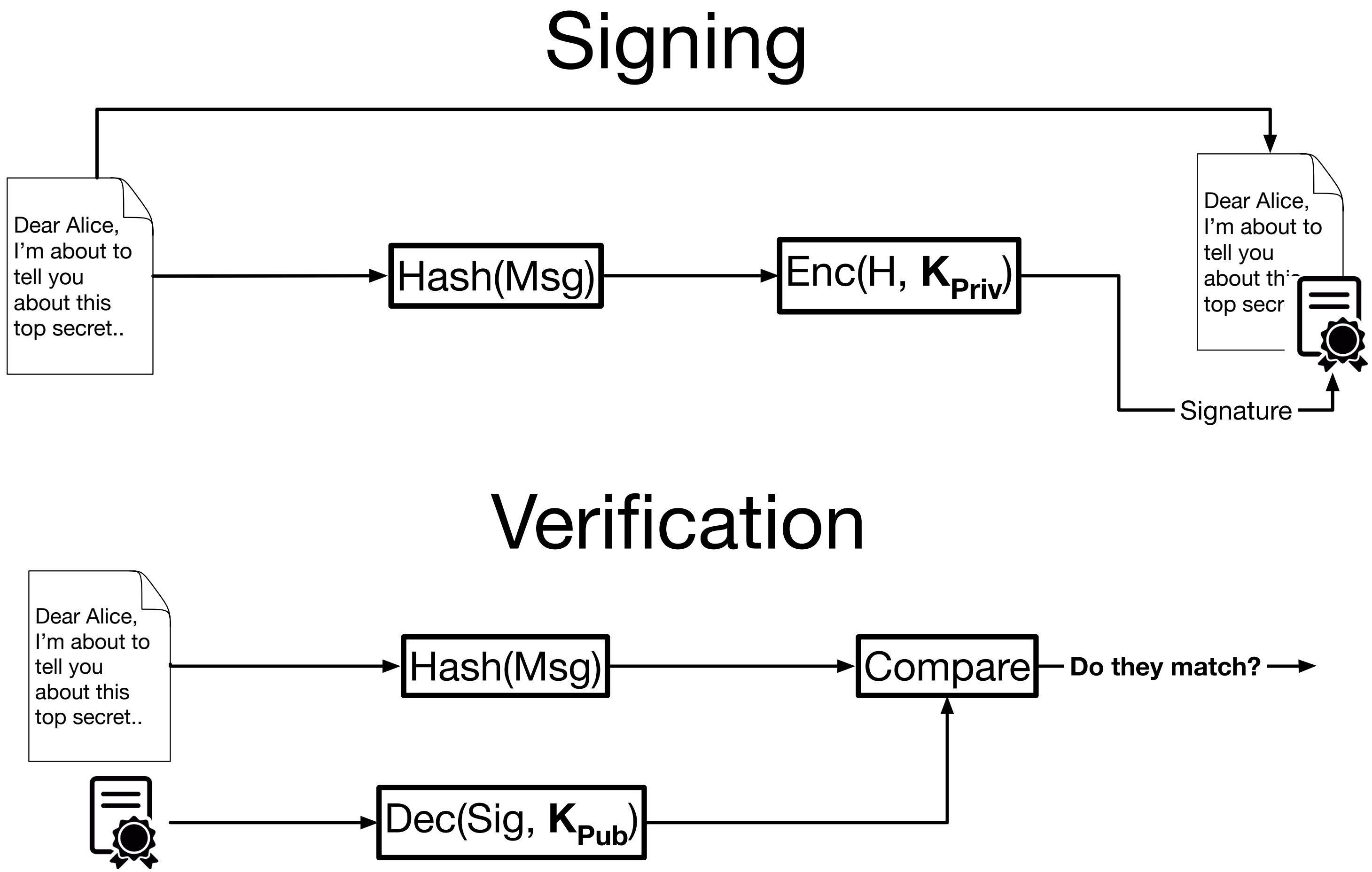
In Public-Key Cryptosystems such as RSA, key generation gives you

- Public Key
  - Used for encrypting data
  - Not a secret
- Private Key
  - Used for decrypting data
  - A secret

Encryption and Decryption



Digital Signatures



SSL/TLS

- Incorporates almost all modern breakthroughs in crypto for our everyday use
- Implements a *hybrid cryptosystem*
  - Key Encapsulation Scheme: adapts public-key cryptosystem to secure the key exchange procedure
  - Data Encapsulation Scheme: adapts efficient symmetric-key cryptosystems for data encryption/decryption
- Ensures Authenticity of server that you are connecting to using digital certicates

  (Public-key cryptosystem)
- Ensures Integrity of the exchanged messages through HMAC

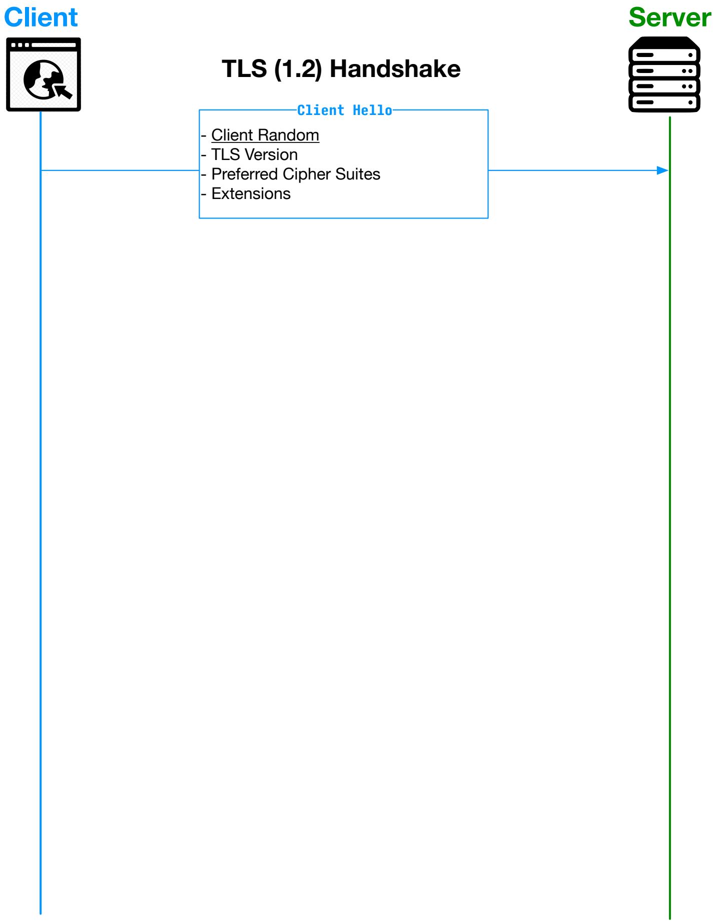
TLS Version and Cipher Suite Selection

TLS_ECDHE-RSA-AES256-GCM-SHA384

ECDHE	RSA	AES256-GCM	SHA384
Key Exchange	Key Authentication	Encryption	Hashing

- Key Exchange: Elliptic-Curve Diffie-Hellman Ephemeral
- Key Exchange Authentication: RSA
- Encryption: AES256-GCM
- Hashing: SHA384

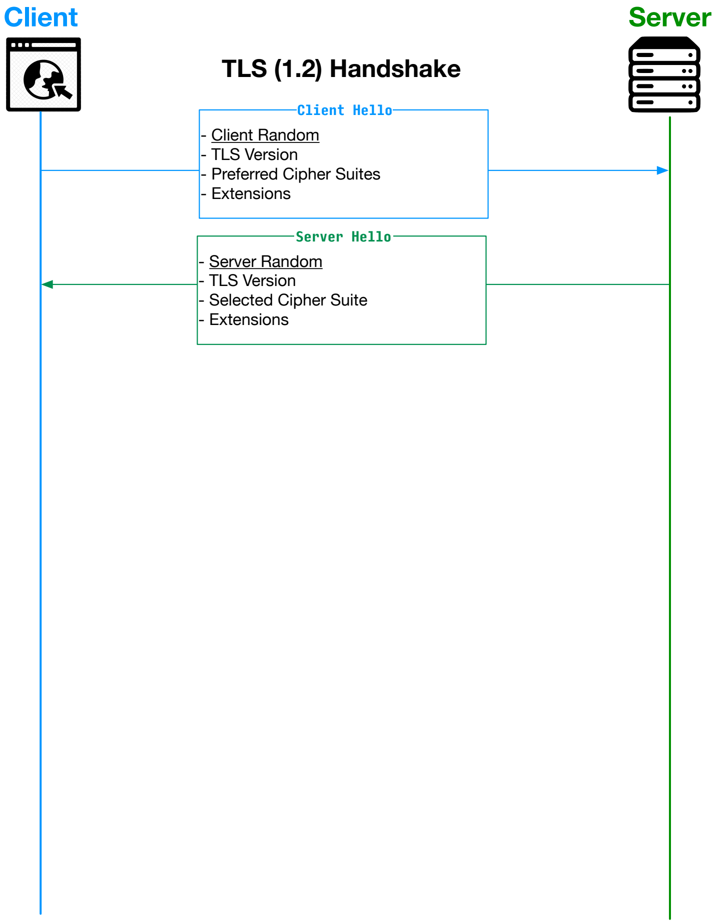
TLS Handshake Illustrated: Client Hello



Client Hello

- Client Random Number R_C
  - Client generates a random number and sends this to server
  - Used for shared secret generation (as we will see shortly)
- Preferred Cipher Suites
  - List of Cipher suites that client supports
- Extensions
  - Extension features defined in TLS

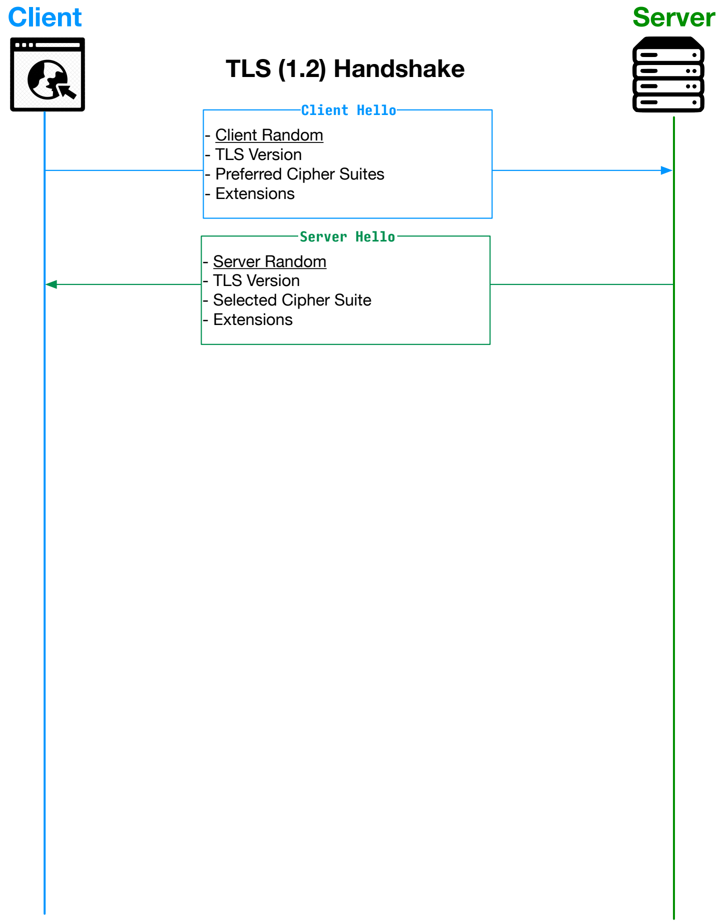
TLS Handshake Illustrated: Server Hello



Server Hello

- Server Random Number R_S
  - Server generates a random number and sends this to Client
  - Used for shared secret generation (as we will see shortly)
- Selected Cipher Suites
  - Server compares its list of cipher suites and client's
  - Makes final selection and notifies
- Extensions
  - Notifies extensions that server supports

TLS Handshake Illustrated: Server Hello

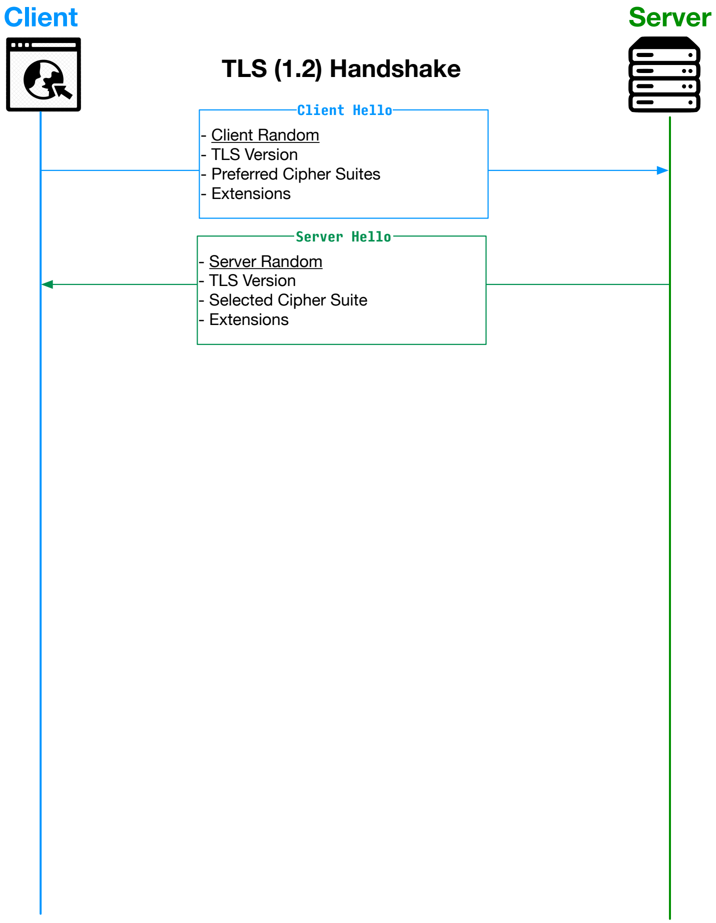


Server Hello

- Cipher Suites Selection
  - Each cipher suites defined in TLS has a code e.g., 0x123 …
  - Server first makes a list of cipher suites supported by both parties
  - Then chooses the strongest (higher code number)

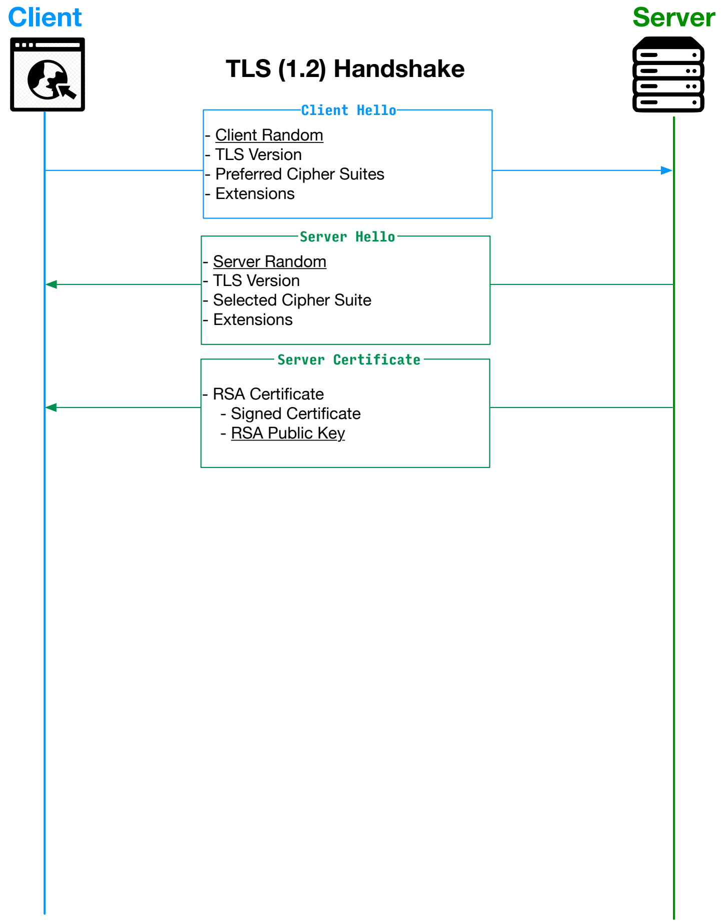
Cipher Suite Name (OpenSSL)	Key Exchange	Encryption	Key Length
ECDHE-RSA-AES256-GCM-SHA384	ECDH 256	AES GCM	256
ECDHE-RSA-AES256-SHA384	ECDH 256	AES	256
ECDHE-RSA-AES256-SHA	ECDH 256	AES	256
DHE-RSA-AES256-GCM-SHA384	DH 1024	AES GCM	256
DHE-RSA-AES256-SHA256	DH 1024	AES	256
DHE-RSA-AES256-SHA	DH 1024	AES	256
DHE-RSA-CAMELLIA256-SHA	DH 1024	Camellia	256
AES256-GCM-SHA384	RSA	AES GCM	256
AES256-SHA256	RSA	AES	256
AES256-SHA	RSA	AES	256
CAMELLIA256-SHA	RSA	Camellia	256
ECDHE-RSA-AES128-GCM-SHA256	ECDH 256	AES GCM	128

TLS Handshake Illustrated: Extensions



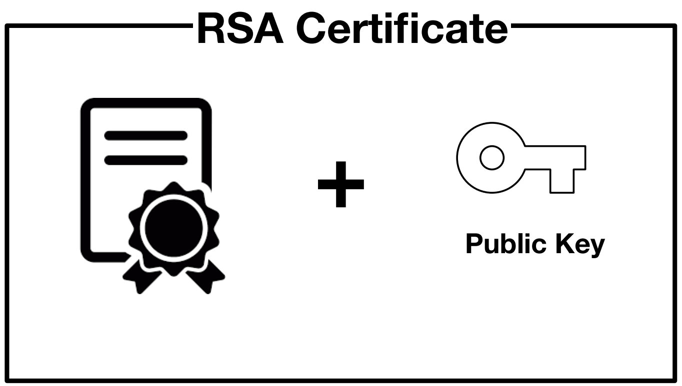
- TLS Extensions
  - Extensions allow certain features to be added *after* the TLS version is standardized
  - Client gives the list of extensions that it wants to use for the session
  - Server sends back *supported extensions*
- Examples of Extensions
  - Session Ticket
    - Allows an established session to be *resumed*
    - Ticket is sent from the server to client for later use
    - Eliminates the need for *renegotiation (handshake)*
  - Server Name Indication (SNI)
    - Allows running multiple SSL/TLS certificates using a same *IP and Port*

TLS Handshake Illustrated: Server Certificate



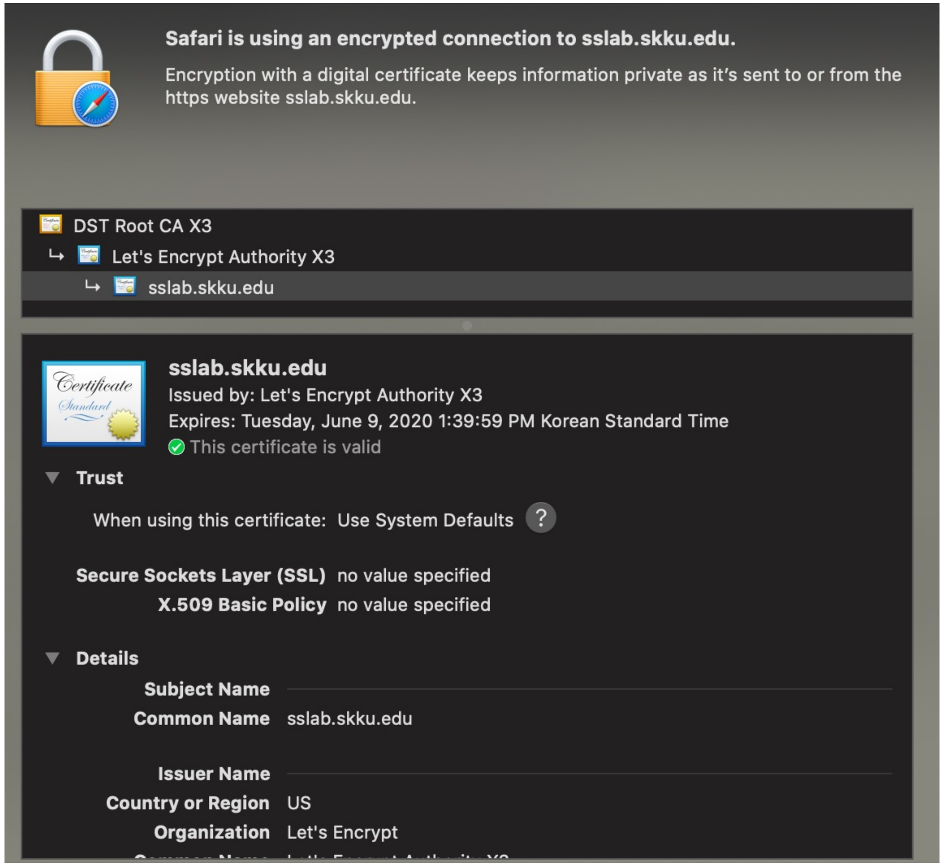
Server Certificate

- Server sends its RSA certificate that can prove its identity
- The certificate consists of 1. signed document + 2. public key for decryption

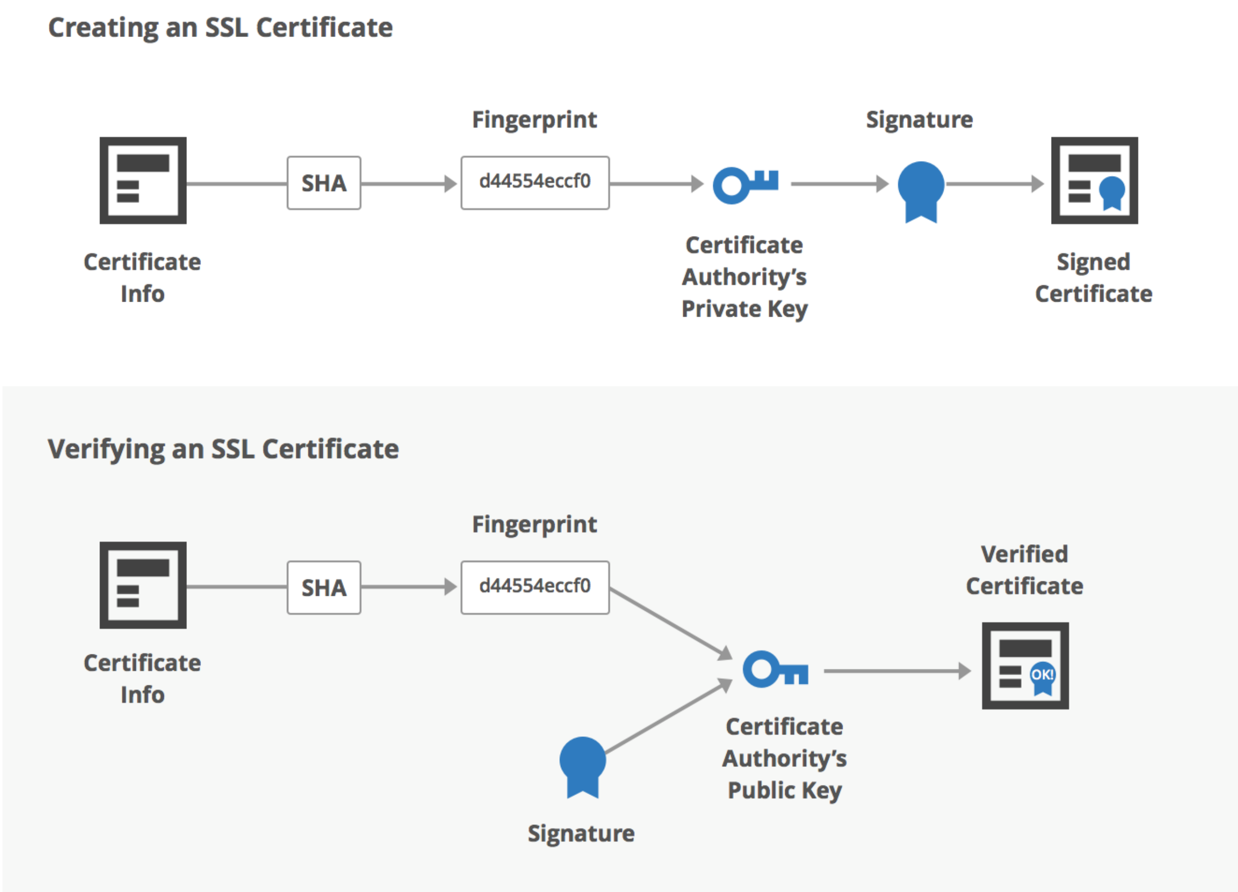


Certificates and Certificate Authorities

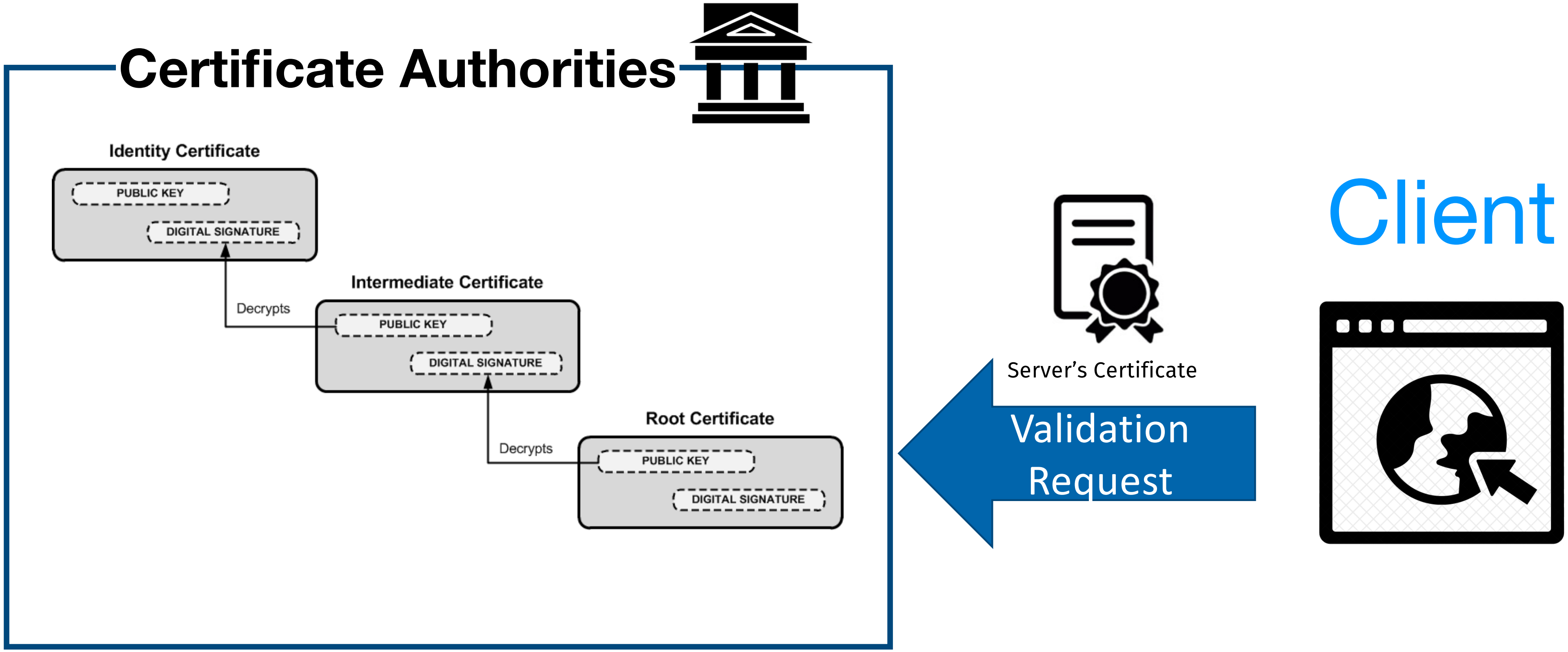
- Certificates <u>authenticate</u> the identity of the server
- Encrypted communication channel ensures <u>confidentiality</u>
- But is the person that you talking to, really that person?



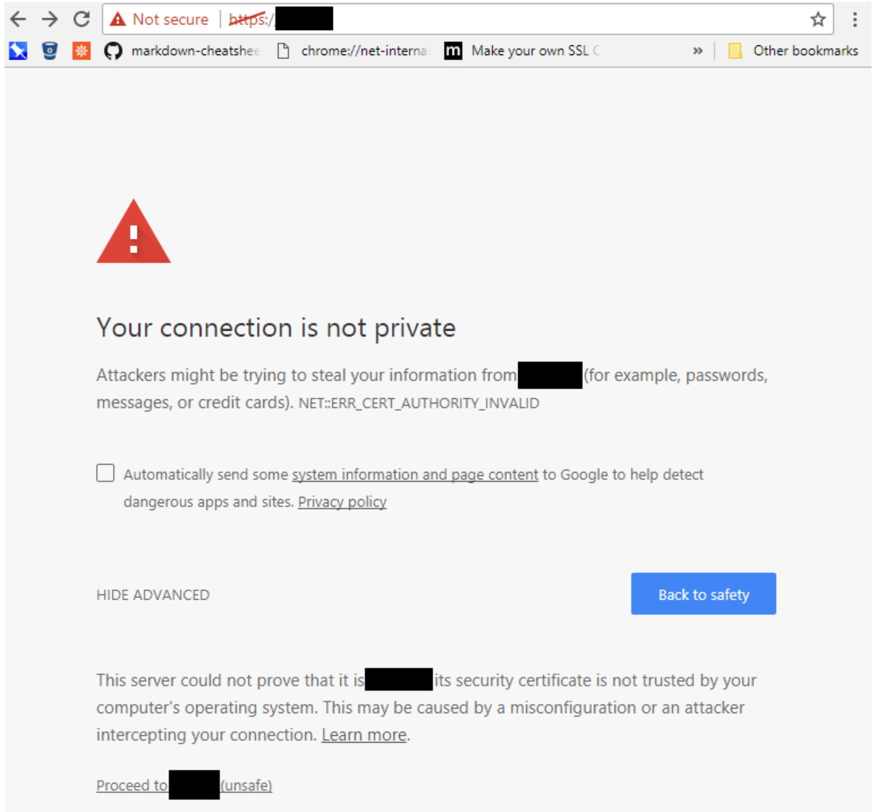
Certificates and Certificate Authorities



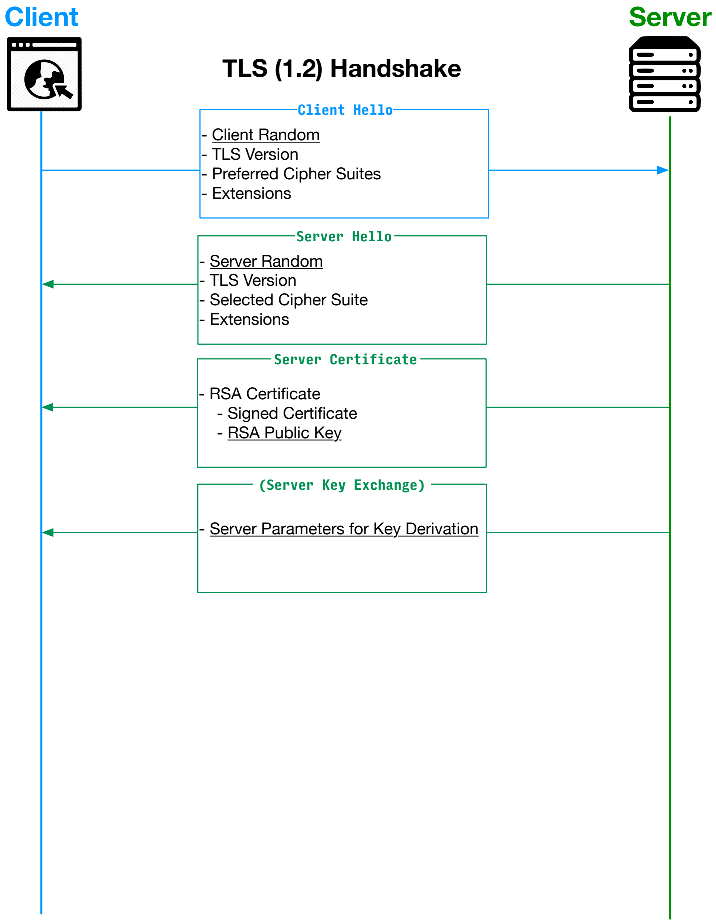
Certificates and Certificate Authorities



Certificates and Certificate Authorities



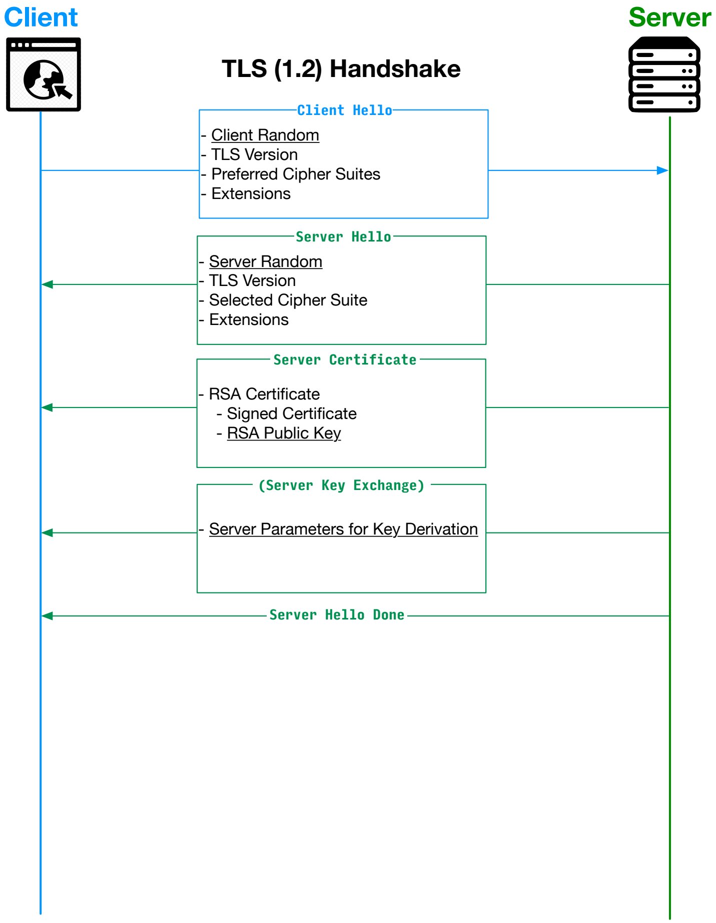
TLS Handshake Illustrated: Server Key Exchange



(Server Key Exchange)

- Server sends parameters that are necessary for session key derivation
- Server Key Exchange
  - is present in cipher suites with Diffie-Hellman
  - is omitted in cipher suites with RSA Key Exchange

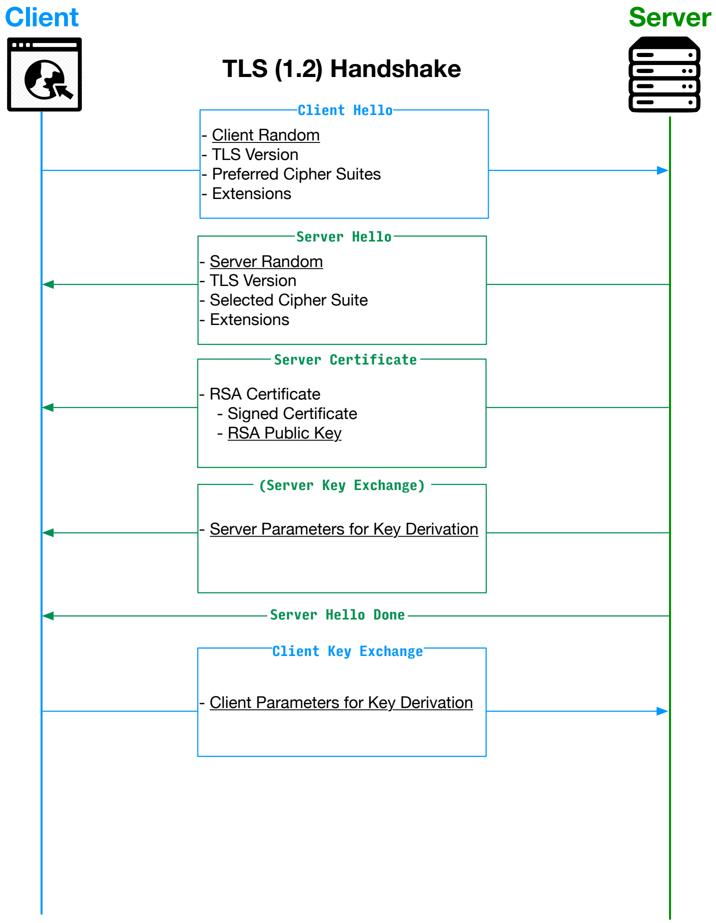
TLS Handshake Illustrated: Server Hello Done



Server Hello Done

- Server's role in the handshake procedure is done.

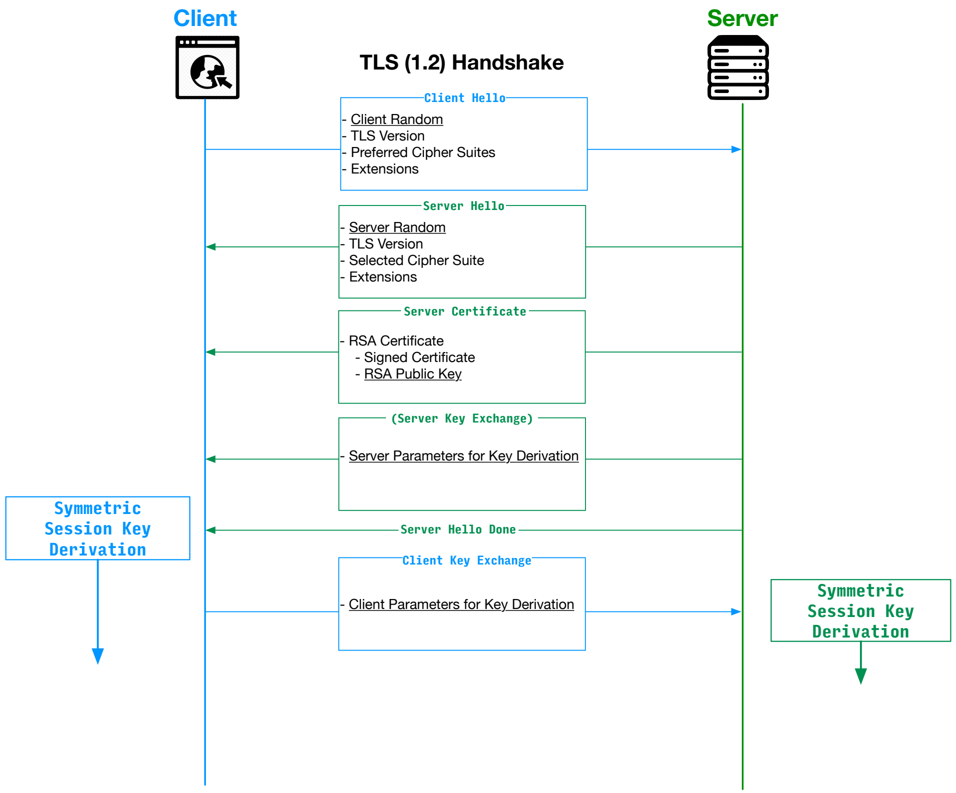
TLS Handshake Illustrated: Client Key Exchange



Client Key Exchange

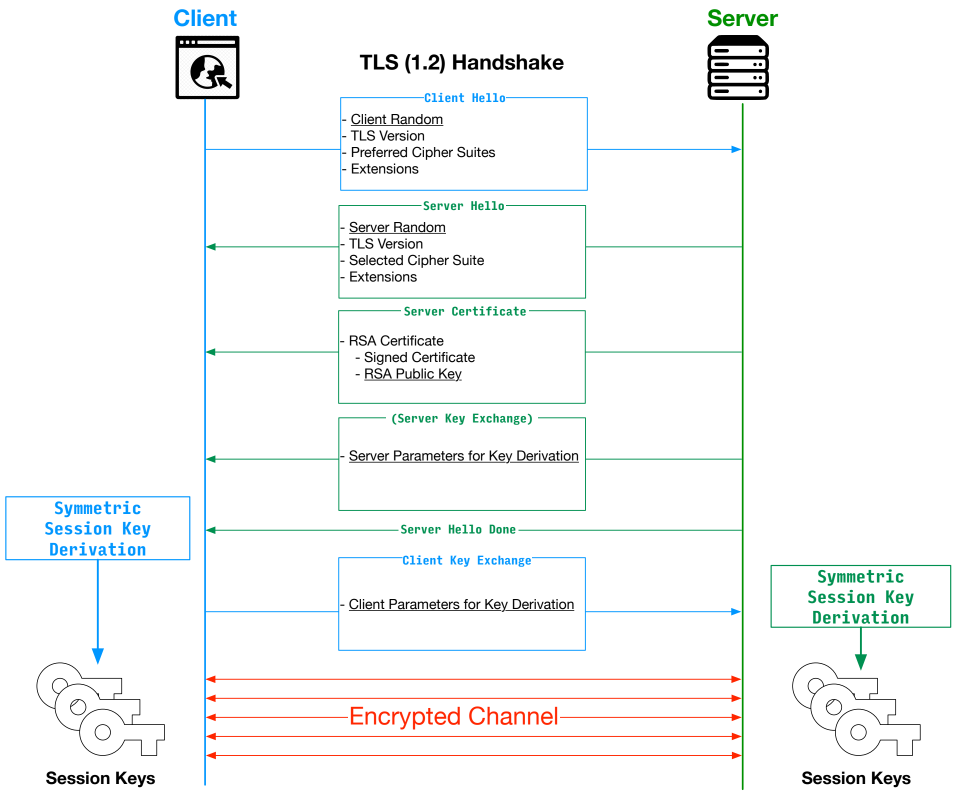
- Client sends parameters that are necessary for session key derivation
- The parameters vary depending on the selected cipher suite
  - In RSA, *Encrypted PreMaster Key is sent*
  - In DH, it's client public key
  - (we'll get to this soon)
- After this point, the both parties have all necessary information for shared secret calculation

TLS Handshake Illustrated: Key Derivation



- Both parties start calculating a <u>shared master secret</u>
- Then derive multiples keys from the master secret e.g.,
  - Key for message
  - Key for HMAC
  - Initialization Vectors for AES

TLS Handshake Illustrated: Done!



- By the magic of key exchange algorithms, both parties exchange public parameters
- And each do their math and end up with the same set of shared asymmetric keys

Software Security Definitions

Memory Safety

Definition: Memory Safety

- Memory Safety is a property that ensure that all memory access adhere to the semantics defined by the source programming language.
- A program is memory safe if all possible execution of that program are *memory safe*

Spatial Memory Safety

Definition: Spatial Memory Safety

- Spatial memory safety is a property that ensure that all memory *dereferences* are within bounds of their pointer's valid objects

- Objects bounds are defined when the object is allocated
  - e.g., `malloc(sizeof(MyObj));`
  - e.g., `char arry[10];`
- Any computed pointer to that object inherits the bounds of the object
  - e.g., `char array[10]; // Bounds &array[0] ~ &array[9]`
  - `char *p = array; // Bounds of p = &array[0] ~ &array[9]`
- Any pointers that point outside of their associated object must not be deferenced
  - `array[11] = ‘a’. // Should not happen`

Spatial Memory Corruption

```
...
char array[10]; // array of 10 chars
array[10] = 'a'; // ???
...
```

- Do you see the bug?
- This is a quintessential case of a spatial memory *bug* that causes memory *corruption*

Temporal Memory Safety

Definition: Temporal Memory Safety

- Temporal memory safety is a property that ensure that all memory dereferences are valid *at the time* of the dereference.

- The object pointed by the pointer is not valid at the time of dereferencing
  - Dereferencing an object that has been freed

Temporal Memory Corruption

```
int* bar(){
    int a = getRandomNumber(); // a = 77;
    int *p = &a;
    return p;
}
void foo(){
    int *p = bar();
    somefunc();
    someOtherfunc(*p);
}
```

- A common mistake I often see C programming beginners
- What is the value of *p?

Temporal Memory Corruption

```
[Thread 1]                  [Thread 3]
...                         obj->studentName;
MyObj* obj =
malloc(sizeof(MyObj));
...

[Thread 2]
...
free(obj);
```

- Use-After-Free: THE most common type of temporal memory corruption
- What if Thread 3 is to call some function of MyObj?

Type Safety

Definition: Type Safety

- Type Safety ensures that only the operations that do not violate the rules of the type system are allowed

Type Safety Violation

```
  struct ObjA{                       struct ObjB{
      int a;                             int a;
      int b;                             int b;
      int c;                         }
  }

ObjA_ptr = (struct ObjA*)& ObjB_instance;
ObjA_ptr->c;  // Totally legal in C
```

- C/C++ does not provide type-safety by design
- Dealing with types and not making errors is up to programmers

Summary: Attacks vs. Defenses

- Attacks
  - Stack code injection
  - Code reuse
  - Memory disclosure
- Defense
  - DEP
  - Stack canary
  - ASLR
  - Etc ..

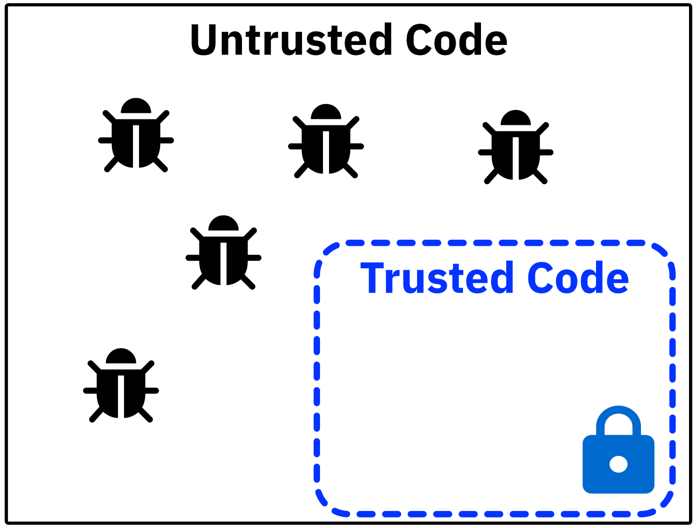
Summary: Attacks vs. Defenses

- Attacks
  - Stack code injection
  - Code reuse
  - Memory disclosure
- Defense
  - DEP
  - Stack canary
  - ASLR
  - Etc ..

Trusted Execution and Cloud

Trusted Execution Approach

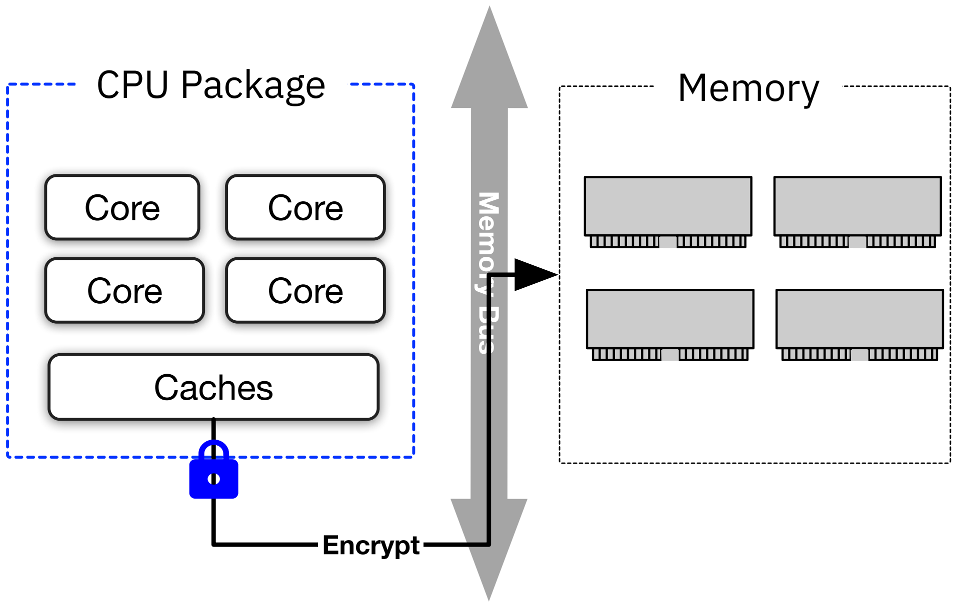
- Regard all SW untrusted
- Create an "*Enclave*" where only small trusted code can be isolated and protected
- Trusted Code Base (TCB)



SGX Architecture: Memory Protection

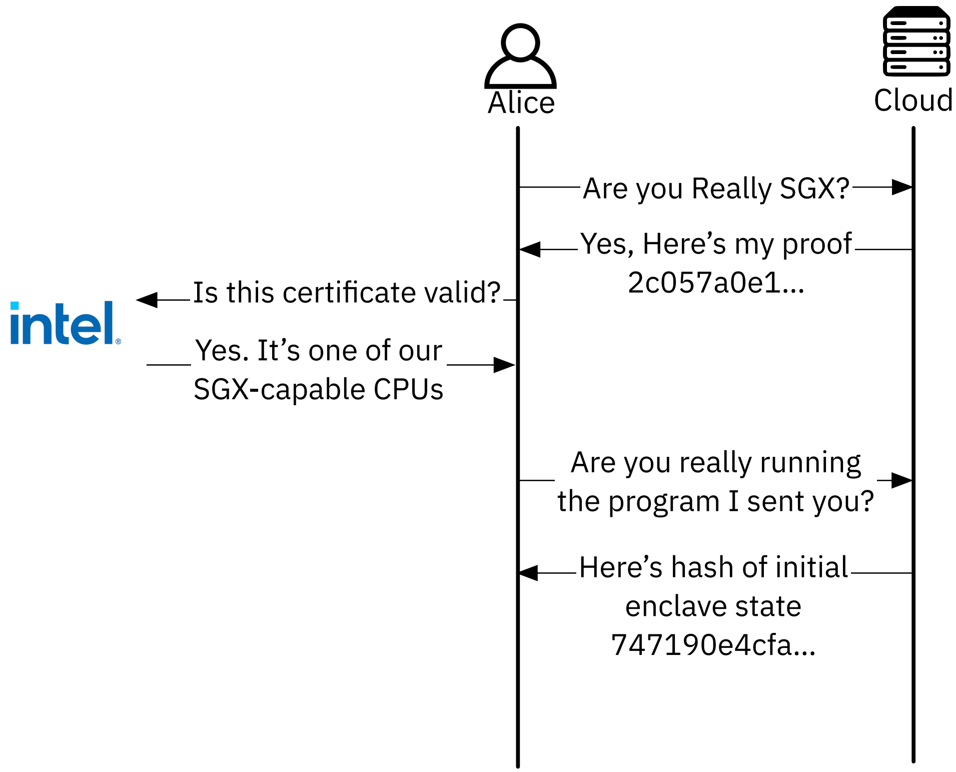
HW Security Perimeter

- Security Perimeter (HW) in SGX is CPU package
- Code/Data are only decrypted inside CPU
- **Snooping on memory bus** is <u>prevented</u>



SGX Security Model: Remote Attestation

Remote Attestation In Human Language



- What we trust
  - 1. CPU
  - 2. Intel (Certificate Authority)
- What we verify
  - 1. It's real SGX
  - 2. It will run our program without modification
- Why is this useful?