Cryptography Simulation with mbedTLS/OpenSSL Library

INSTRUCTORS: VIKASH MISHRA, VIDHYA KRISHNAMURTHY

intel®

Intel Mentors

- Vikash Mishra -*“experienced Security Research Engineer/Scientist and Product Security Expert at Intel. With over 13 years of experience in the Security domain, Vikash's expertise spans a wide range of areas, including Cryptography, Cloud Security, System Security, Web Security, and Network Security etc. At Intel, his primary responsibility is to ensure that products undergo a rigorous Secure Development Lifecycle, which encompasses activities such as threat modeling, cryptography review, secure code review, etc. and the execution of advanced SAST and DAST tools to maintain the highest security standards. His methodical approach to security analysis and dedication to safeguarding Intel's products positions him as a leading figure in the field of security research and development.”*

- Vidhya Krishnamurthy – “*coming from an embedded background, Vidhya has been working on security for the past few years. Her expertise spans from embedded to cloud technologies. At Intel she is a Security researcher with the primary task of securing Intel products. She also owns the Intel India site level security initiatives resulting in growing security competence at the site. Her interests include confidential computing, encompassing cryptography and secure execution. As an instructor of security and cryptography classes at Intel, she is very keen to educate interested people in this space.* ”

Cryptography ?

Why should you use cryptography?

- Cryptography can be used to implement part of security mitigations for identified threats
- Threat modeling process defines what assets (and CIA* properties) need protection
- Based on identifying adversaries and threats
  - Identity **spoofing**
  - Data **tampering**
  - **Repudiation**
  - Unauthorized **information** disclosure
  - **Denial** of service
  - Unauthorized **elevation** of privileges

* CIA – Confidentiality, Integrity, Availability

Crypto can provide

- Confidentiality – The data cannot be read by anyone else
- Authenticity – The receiver/reader knows that data originated from a trusted source
- Integrity – The data hasn't been tampered with in transit/storage
- Non-repudiation – The sender cannot dispute its authorship or the validity of a data

What will you learn in this training

- Practice using recommended **cryptographic algorithms**
- Familiarize through practice with popular cryptographic libraries APIs
- Practice to avoid insecure **coding practices** with regards to using cryptography

Course Outline

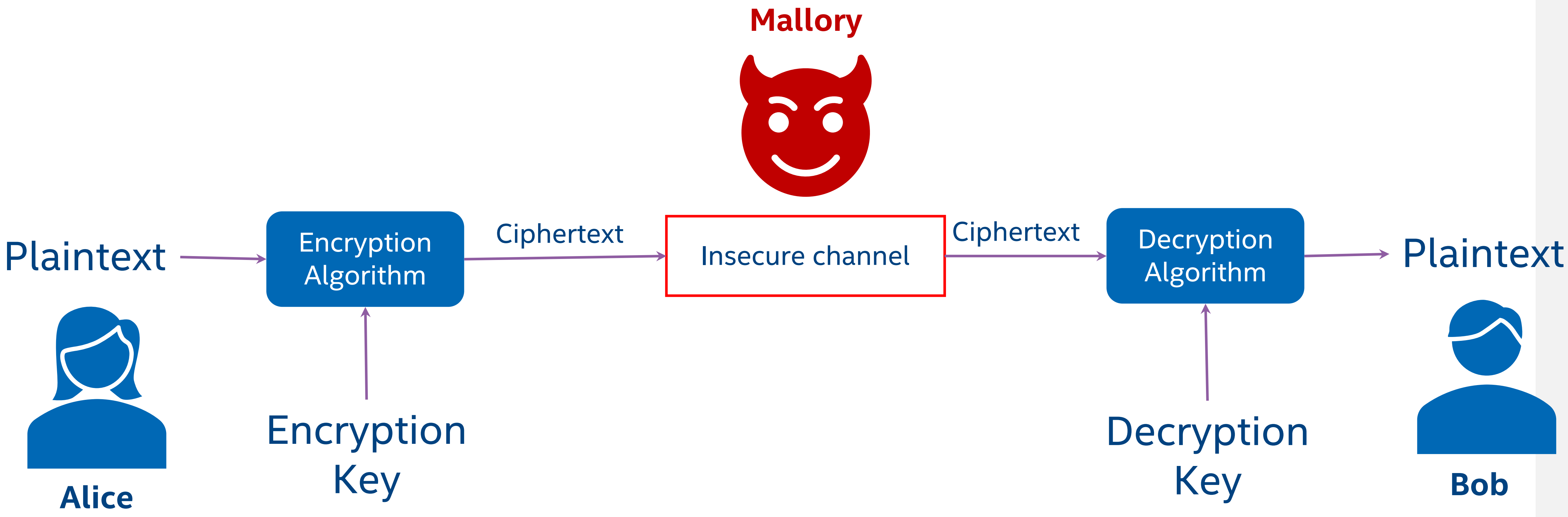
1. Introduction to Cryptography Algorithm
2. Exercise #1 – Creating Digital Certificates
3. Exercise #2 – Securing a custom protocol
4. Summary

Disclaimer

- The following scenarios are designed to show you basic concepts but is not designed to be used for production as is.

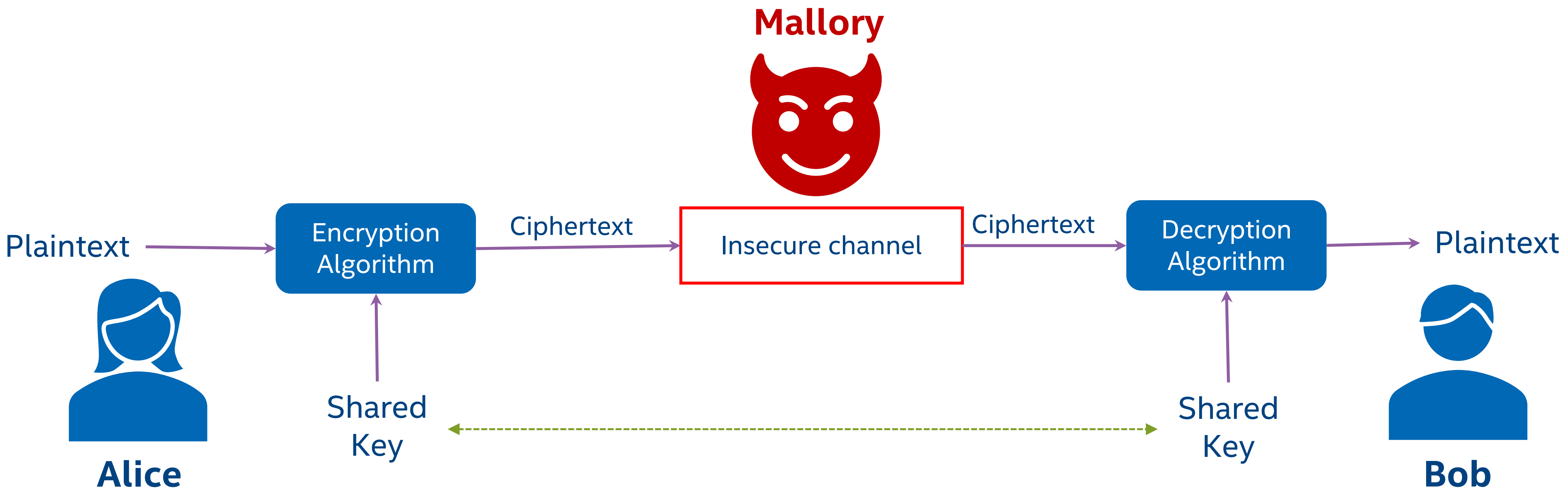
CRYPTOGRAPHY ALGORITHMS

Basic Cryptosystem



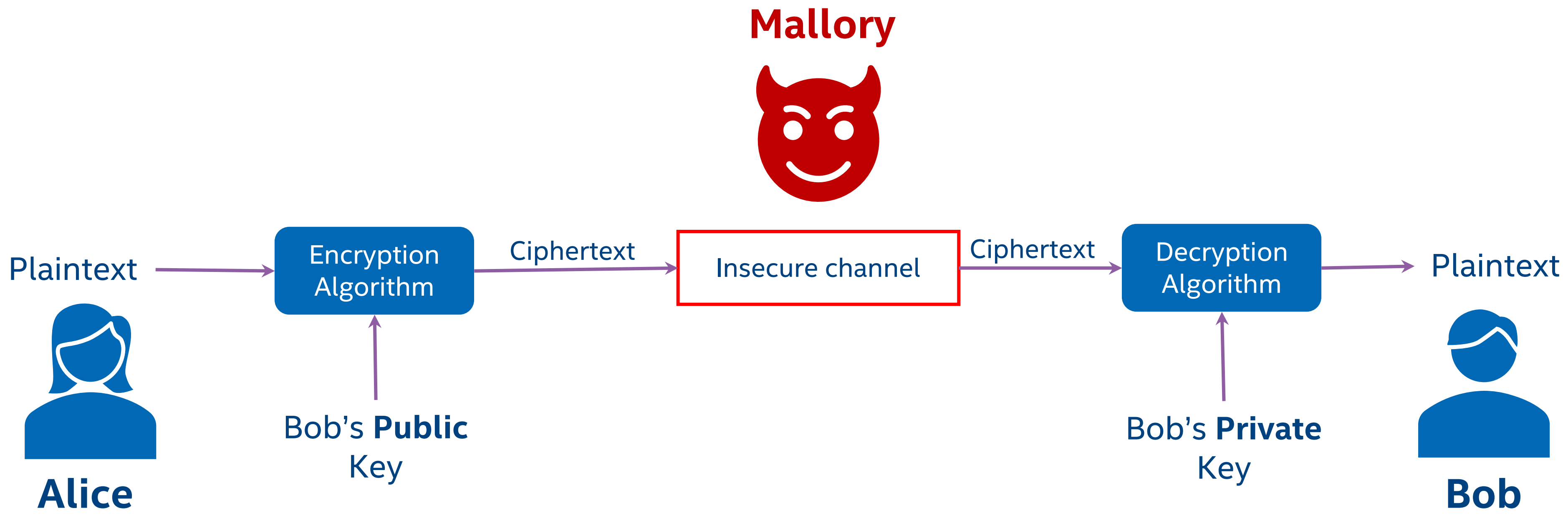
Basic Cryptosystem - Symmetric key cryptography

- Same key is shared between parties (how?)
- Cipher text ≈ same length as plaintext
- Examples: AES, 3DES, SM4



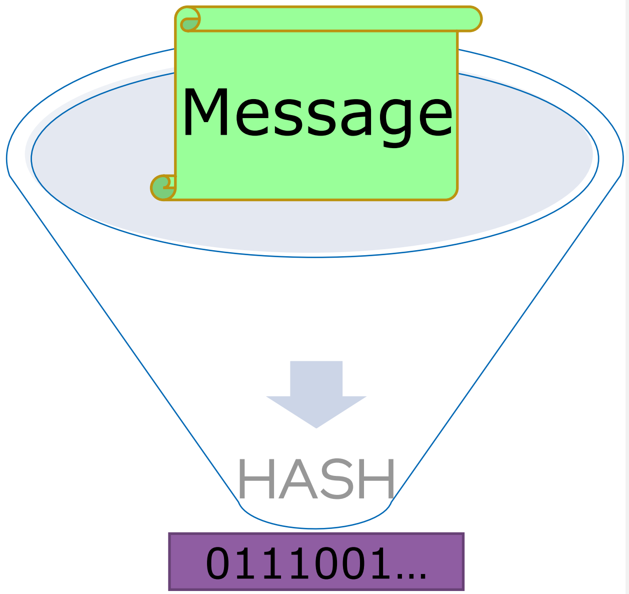
Basic Cryptosystem - Asymmetric key encryption

- Also known as "Public key cryptography"
- Key has 2 mathematically related parts: private (secret), public (shared)
- Much slower than symmetric key cryptography
- Examples: RSA encrypt/decrypt (Publicly invented in 1975), SM2

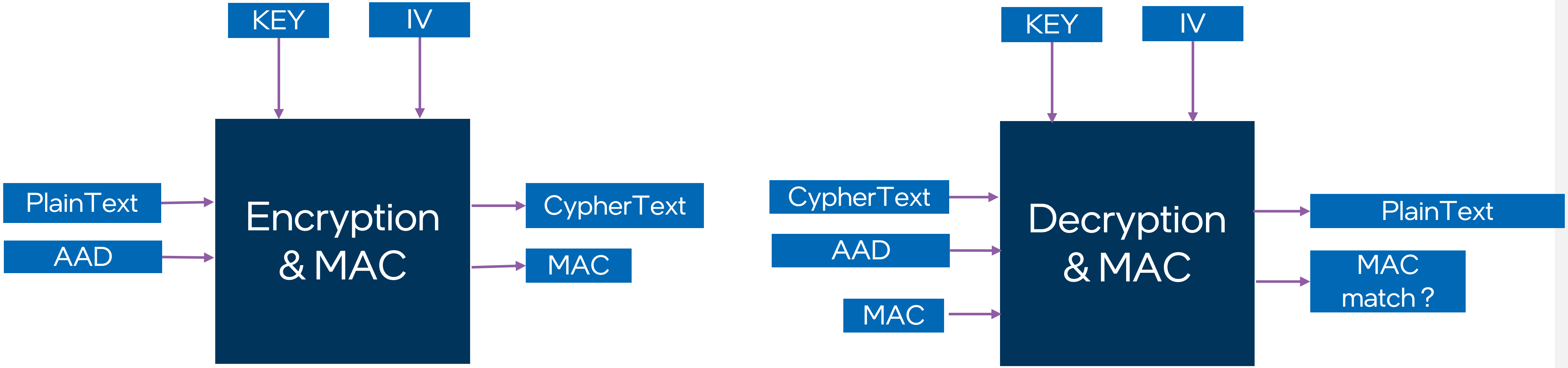


Cryptographic hash

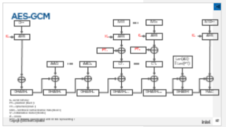
- Maps data of arbitrary size to a bit string of a fixed size
- No keys!
- Example: ~~SHA1~~, ~~MD5~~, SHA2 (SHA~~256~~/384/512), SM3, SHA3



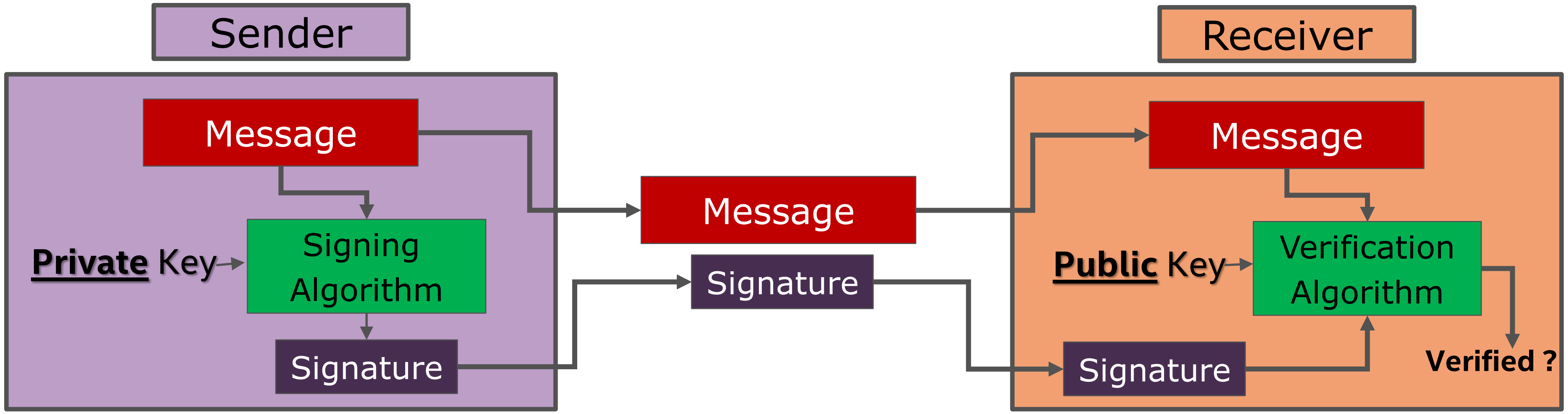
AES-GCM - Galois/Counter Mode



- Combined Encryption and MAC with same operation using the same key - uses CTR mode
- It's a non-deterministic mode - changing IV → different ciphertext and MAC for the same plaintext (IV is not secret)
- Never repeat same IV and Key combination
- Additional Authenticated Data (AAD) is not encrypted but used for MAC generation
- Can provide authentication only by using AAD only → GMAC
- Ciphertext size == plaintext size (no need for padding)



Digital Signatures

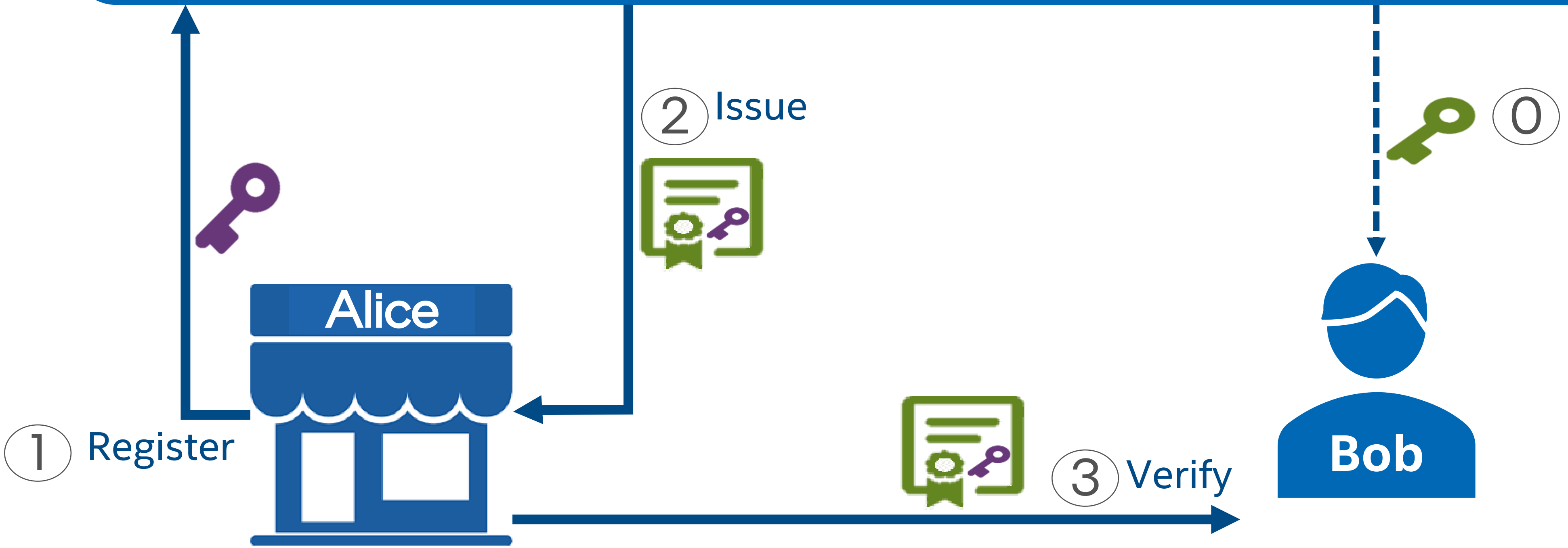


- **Provides:**
  - **Authenticity** – The message came from the stated sender
  - **Integrity** – The message has not been changed
  - **Non-repudiation** - The sender cannot dispute its authorship
- A successful authentication guarantees that the message was signed by the owner of the private key that corresponds to the public key found in the certificate.

Public Key Infrastructure

Certificate Authority

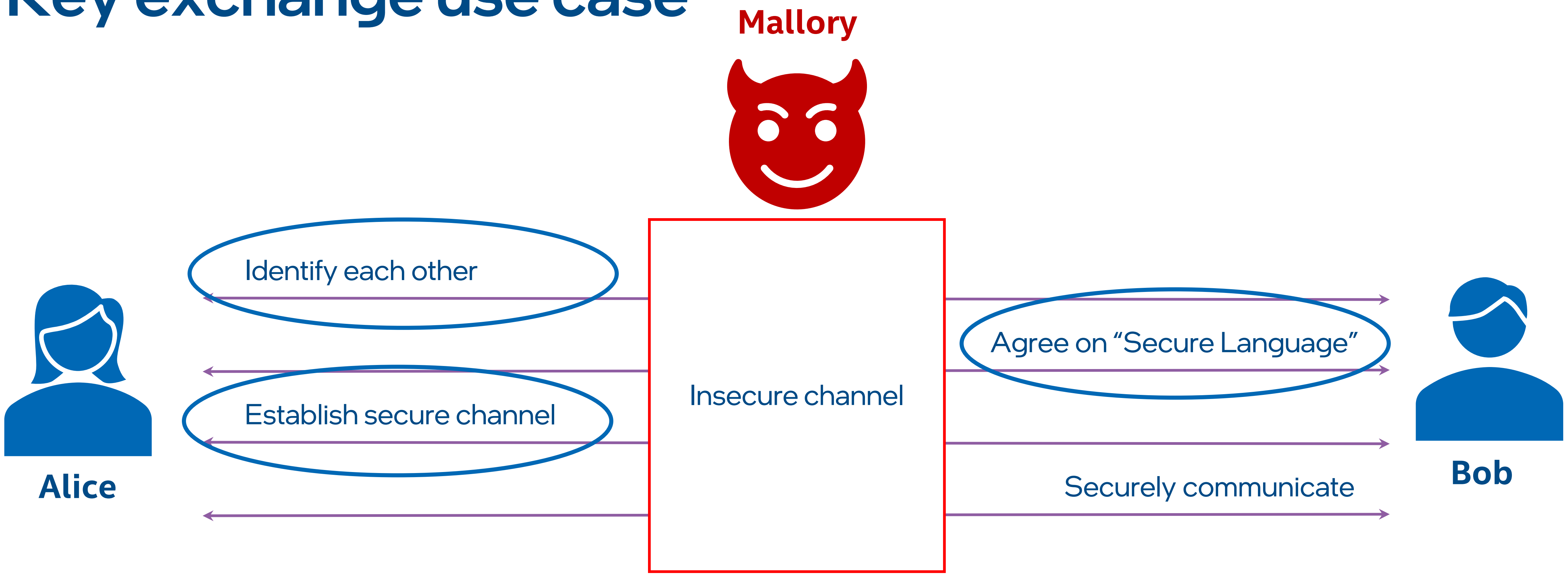
- **Register new users** (Registration Authority)
- **Generates and publishes certificates** (Certificate Authority)
- **Verifies certificates** (Verification Authority)



KDF - two phases approach

- Phase 1 - Entropy extraction phase
  - To address the gaps of the initial secret
  - Applied to a weakly random entropy source, together with a salt to generate a highly random output that appears independent from the source and uniformly distributed
- Phase 2 – Key expansion phase
  - Generate keys from the Entropy extraction phase output and key context
  - Each key is independent from the input and the other generated keys
  - Examples: Different key for different purpose, or different key per new version

Key exchange use case



What part of the protocol is key exchange?

Key Exchange Requirements

1. Establish a shared secret between two parties
2. Authenticate the other party
3. Forward secrecy

Diffie-Hellman

Digital signatures

Ephemeral keys

What cryptographic methods can you use for each requirement?

Diffie-Hellman principles - mathematical difficulty

- Based on Discrete Logarithm Problem mathematical difficulty
- Generalized to any finite cyclic group, e.g. 0,1,..., *p*-1 (p is a prime) or points on a discrete Elliptic Curve

Given *p, g and y, such as* ***y = g*** ^{x} ***mod p*** **–** find *x*

$$g, g^2, g^3, \ldots, \underbrace{g^{340282366920938463463374607431768211456}}_{\textbf{DH Private Key}}, \ldots \bmod p$$

Diffie-Hellman principles – shared secret

- DH shared secret calculation is based on the commutativity property
  - Shared secret = $g^{xy} = (g^x)^y = (g^y)^x$
- If each party knows own private key and the other's party public key – it can calculate the shared secret.
- Mallory can't calculate the shared secret from g^x and g^y

	Party A knowledge	Party B knowledge	MiM knowldge
Private Key	x	y	-
Public Key	g^x	g^y	-
Other party public key	g^y	g^x	g^y, g^x
Shared Secret	$g^{xy} = (g^y)^x$	$g^{xy} = (g^x)^y$	$g^{xy} = ?$

Diffie-Hellman example

Alice and Bob agreed upfront on group parameters:
$p = 13$ (modulus)
$g = 6$ (generation element)

Alice

Choose random private key: x = 5
Calculate public: $g^x \bmod p = 6^5 \bmod 13 = 2$
Send: Alice, 2

Bob, 9

Shared secret: $(g^y)^x \bmod p = 9^5 \bmod 13 = 3$

Insecure channel

Mallory

Alice, 2

Bob

Choose random private key: y = 4
Calculate public: $g^y \bmod p = 6^4 \bmod 13 = 9$
Send: Bob, 9

Shared secret: $(g^x)^y \bmod p = 2^4 \bmod 13 = 3$

Never use the shared secret as a cryptographic key!
Always use a standard Key Derivation Function (KDF) to derive cryptographic keys from a shared secret!

Diffie-Hellman example

Alice and Bob agreed upfront on:
$p = 13$ (modulus)
$g = 6$ (generation element)

Choose random private key: x = 5
Calculate public: $g^x \bmod p = 6^5 \bmod 13 = 2$
Send: Alice, 2

Alice

Bob, 9

Shared secret: $(g^y)^x \bmod p = 9^5 \bmod 13 = 3$

Insecure channel

Mallory

Alice, 2

Choose random private key: y = 4
Calculate public: $g^y \bmod p = 6^4 \bmod 13 = 9$
Send: Bob, 9

Shared secret: $(g^x)^y \bmod p = 2^4 \bmod 13 = 3$

Bob

Can we use this protocol for establishing a secure channel? Why?

Diffie-Hellman identity problem

Mallory

Choose random private: x = 5
Calculate public: $g^x \ mod \ p = 6^5 \bmod 13 = 2$
Send: Alice, 2

Choose random private key:
$x' = 2 \rightarrow g^{x'} \ mod \ p = 10$

Alice, 10

Choose random private key:
$y' = 3 \rightarrow g^{y'} \ mod \ p = 8$

Choose random private: y = 4
Calculate public: $g^y \ mod \ p = 6^4 \bmod 13 = 9$
Send: Bob, 9

Insecure channel

Bob, 8

Shared key with Alice:
$2^3 \bmod 13 = 8$

Shared secret: $(g^y)^x \bmod p = 8^5 \bmod 13 = \mathbf{8}$

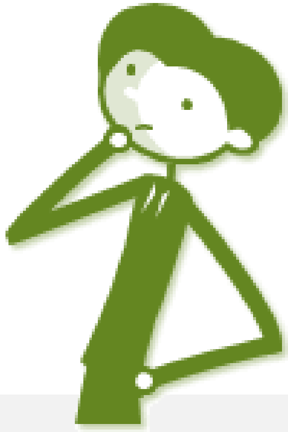
Shared secret: $(g^x)^y \ mod \ p = 10^4 \bmod 13 = \mathbf{3}$

Alice

Bob

Shared key with Bob:
$9^2 \bmod 13 = 3$

How would you solve this Man In the Middle problem?



intel.

Diffie-Hellman identity problem solution

- The solution has to cryptographically “bind together” the message parts
- Seems like a simple task
- Several attempts were done
- Each had mistakes, eventually leading to SIGn and MAc (SIGMA) definition

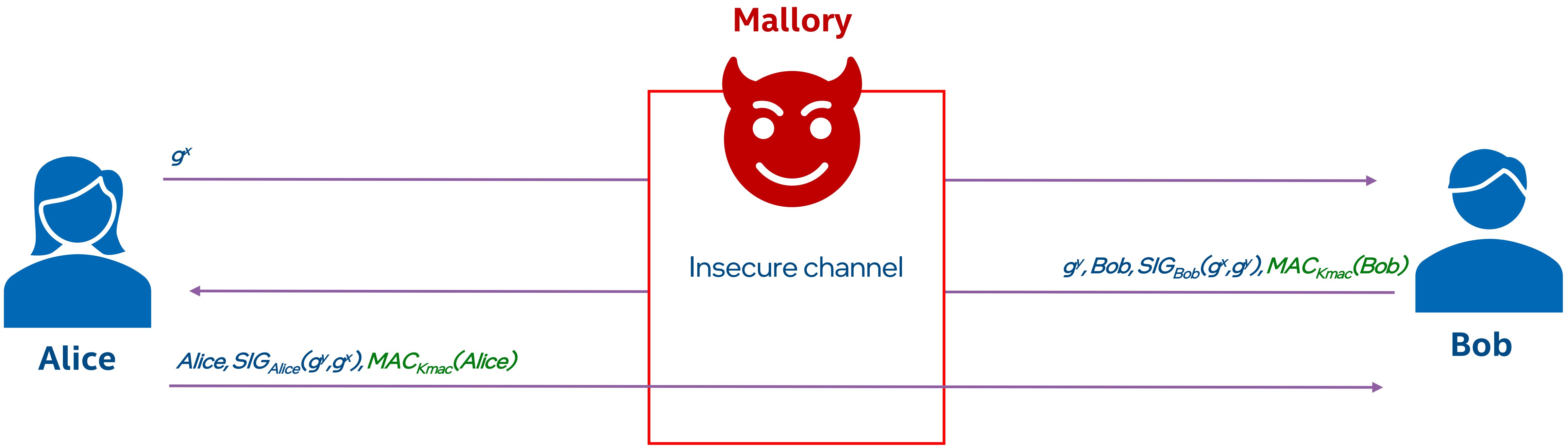
SIGMA took the good and learned from the bad of previous protocols

SIGMA

- Need to bind the derived keys (K) with the peer identities
- SIGn and MAc
  - Sign the two ephemeral public keys with own identity
  - MAC own identity with a key derived from the shared secret

SIGMA – basic version

$$K_{mac} = KDF_{mac}(g^{xy})$$

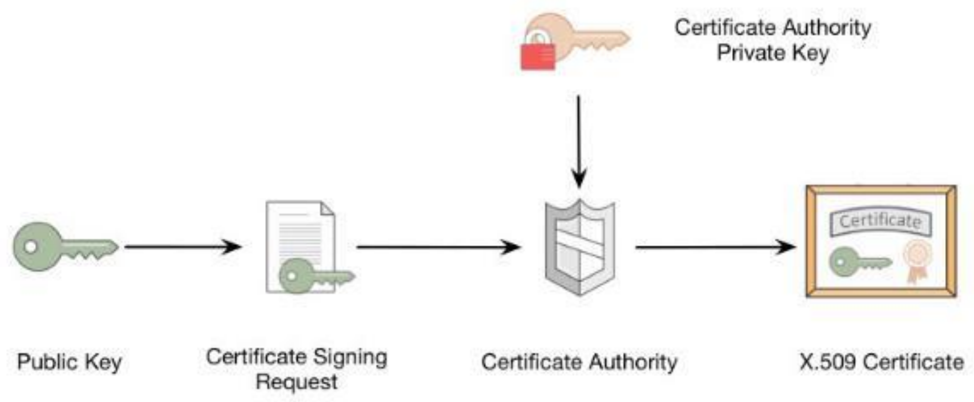


Theory and practice

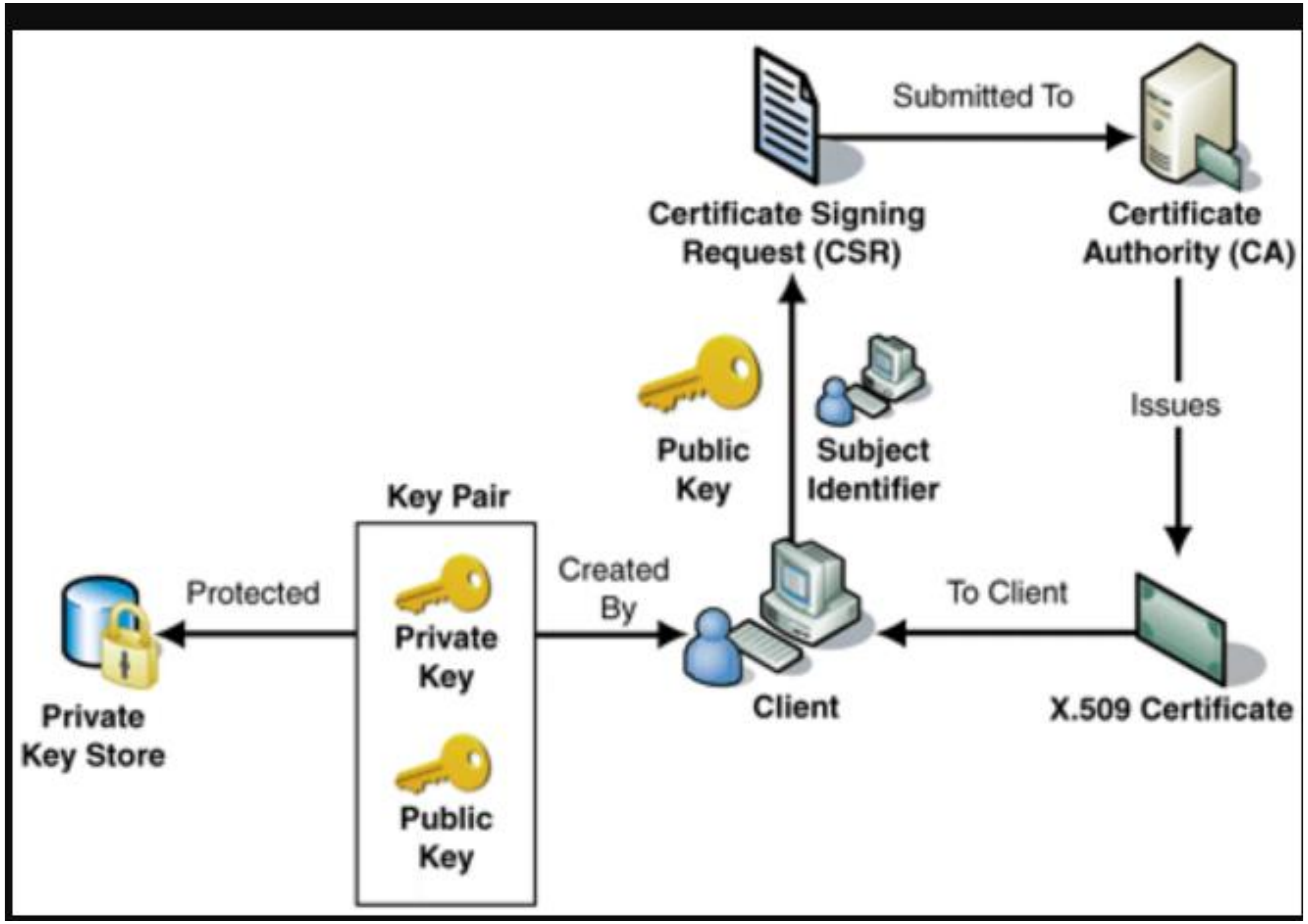
Exercise #1 – Creating Digital Certificates

Certificate Signing Request

- A certificate signing request (CSR) is a message sent to a certificate authority to request the signing of a public key and associated information
- Most commonly a CSR will be in a PKCS10 format
- The contents of a CSR comprises a public key, as well as a common name, organization, city, state, country, and e-mail
- Not all these fields are required and will vary depending on the assurance level of your certificate.



Certificate creation flow



CSR fields

DN[1]	Information	Description	Sample
CN	Common Name	This is fully qualified domain name that you wish to secure	*.wikipedia.org
O	Organization Name	Usually the legal name of a company or entity and should include any suffixes such as Ltd., Inc., or Corp.	Wikimedia Foundation, Inc.
OU	Organizational Unit	Internal organization department/division name	IT
L	Locality	Town, city, village, etc. name	San Francisco
ST	State	Province, region, county or state. This should not be abbreviated (e.g. West Sussex, Normandy, New Jersey).	California
C	Country	The two-letter ISO code for the country where your organization is located	US
EMAIL	Email Address	The organization contact, usually of the certificate administrator or IT department	

CSR Format

- The CSR itself is usually created in a Base-64 based PEM format.
- You can open the CSR file using a simple text editor and it will look like the sample below.
- You must include the header and footer (-----BEGIN NEW CERTIFICATE REQUEST-----) when pasting the CSR.


```
-----BEGIN CERTIFICATE REQUEST-----
MIICzDCCAbQCAQAwgYYxCzAJBgNVBAYTAkVOMQ0wCwYDVQQIDARub25lMQ0wCwYD
VQQHDARub25lMRIwEAYDVQQKDAlXaWtpcGVkaWExDTALBgNVBAsMBG5vbmUxGDAW
BgNVBAMMDyoud2lraXBlZGlhLm9yZzEcMBoGCSqGSIb3DQEJARYNbm9uZUBub25l
LmNvbTCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBAMP/U8RlcCD6E8AL
PT8LLUR9ygyygPCaSmIEC8zXGJung3ykElXFRz/Jc/bu0hxCxi2YDz5IjxBBOpB/
kieG83HsSmZZtR+drZIQ6vOsr/ucvpnB9z4XzKuabNGZ5ZiTSQ9L7Mx8FzvUTq5y
/ArIuM+FBeuno/IV8zvwAe/VRa8i0QjFXT9vBBp35aeatdnJ2ds50yKCsHHcjvtr
9/8zPVqqmhl2XFS3Qdqlsprzbgksom67OobJGjaV+fNHNQ0o/rzP//Pl3i7vvaEG
7Ff8tQhEwR9nJUR1T6Z7ln7S6cOr23YozgWVkEJ/dSr6LAopb+cZ88FzW5NszU6i
57HhA7ECAwEAAaAAMA0GCSqGSIb3DQEBBQUAA4IBAQBn8OCVOIx+n0AS6WbEmYDR
SspR9xOCoOwYfamB+2Bpmt82R01zJ/kaqzUtZUjaGvQvAaz5lUwoMdaO0X7I5Xfl
sllMFDaYoGD4Rru4s8gz2qG/QHWA8uPXzJVAj6X0olbIdLTEqTKsnBj4Zr1AJCNy
/YcG4ouLJr140o26MhwBpoCRpPjAgdYMH60BYfnc4/DILxMVqR9xqK1s98d6Ob/+
3wHFK+S7BRWrJQXcM8veAexXuk9lHQ+FgGfD0eSYGz0kyP26Qa2pLTwumjt+nBPl
rfJxaLHwTQ/1988G0H35ED0f9Md5fzoKi5evU1wG5WRxdEUPyt3QUXxdQ69i0C+7
-----END CERTIFICATE REQUEST-----
```

Exercise

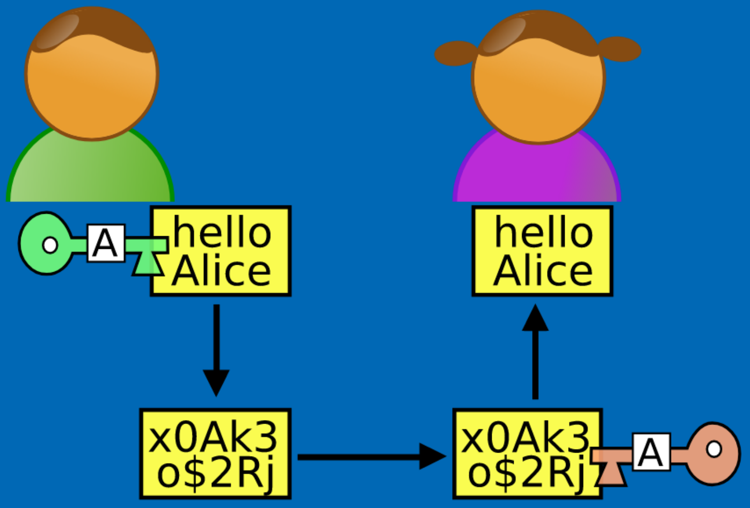
Creating Digital Certificates

1. Create a Self Signed root certificate(rootCA.crt) with RSA key size of 3072 with SHA384 and set serial number 01
2. Generate RSA keypair of size 3072 with SHA384 for “Alice” and sign with root CA and set serial number 02
3. Generate RSA keypair of size 3072 with SHA384 for “Bob” and sign with root CA and set serial number 03

```
openssl x509 -text -noout -in cert_file.crt
```

Exercise #2 - Securing a custom protocol

Putting your crypto knowledge to practice



Exercise - Securing a custom protocol

- You will be securing a custom communication protocol between two parties

A patient

An "AI" psychologist - Eliza

- Your job – to secure the traffic
- Two phases
  1. Basic - Implement crypto wrapper and make the crypto unit-test pass
  2. Advanced - Secure the protocol using your crypto wrapper implementation

Usage

Client

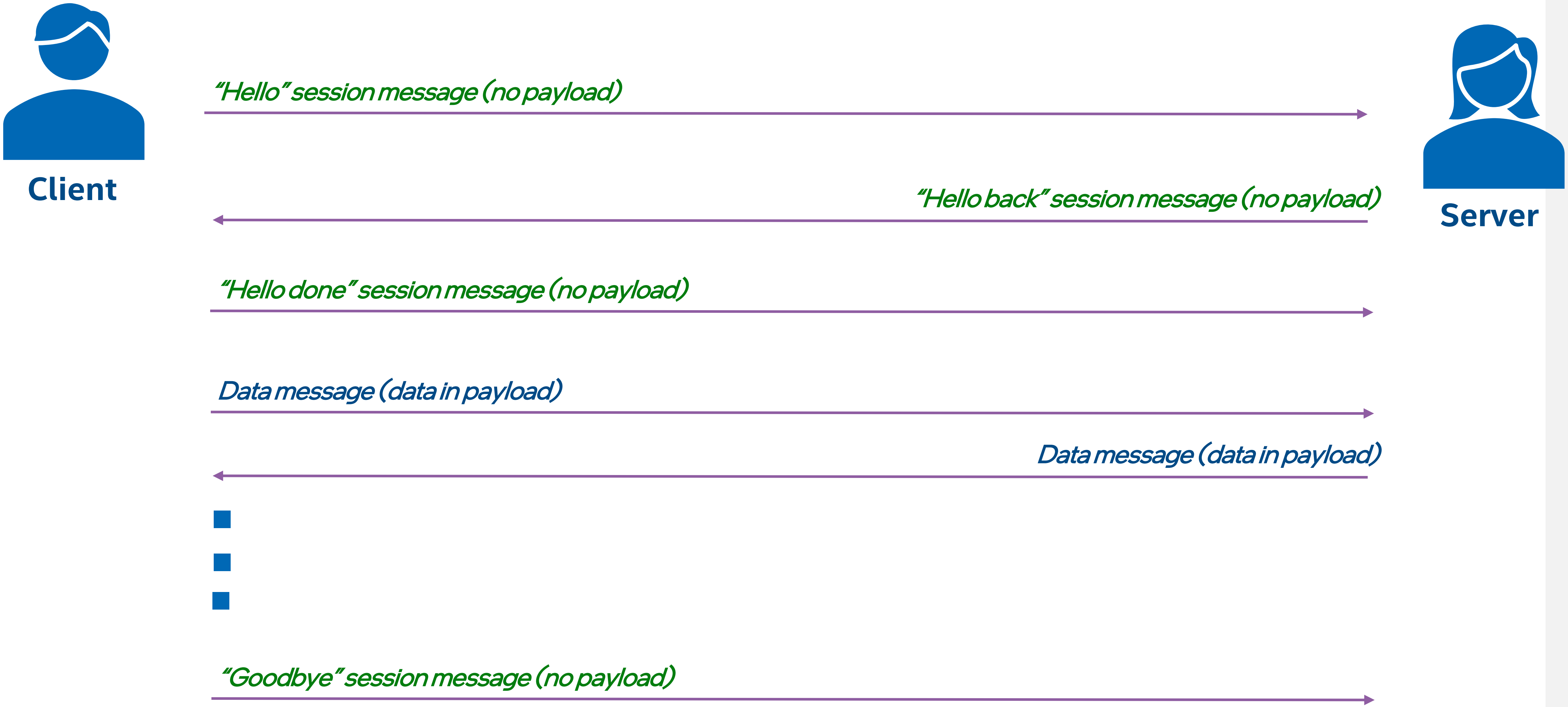
```
C:\Users\alexber\OneDrive - Intel Corporation\Documents\GitHub\documentation.training.applied-crypto-part2\udp_party\x64
\Debug>udp_party -ip 127.0.0.1 -port 3000 -key bob.key -pwd bobkey -cert bob.crt -root rootCA.crt -peer Alice.com
Session started with Alice.com
Received response:"HI!  I'M ELIZA.  WHAT'S YOUR PROBLEM?"
I'm trying to write crypto code
Received response:"DID YOU COME TO ME BECAUSE YOU ARE TRYING TO WRITE CRYPTO CODE?"
yes
Received response:"YOU SEEM QUITE POSITIVE."
Writing crypto code is difficult
Received response:"SAY, DO YOU HAVE ANY PSYCHOLOGICAL PROBLEMS?"
anyone who writes crypto code has problems
Received response:"WHY DO YOU ASK?"
are you real?
Received response:"WHY ARE YOU INTERESTED IN WHETHER OR NOT I AM REAL?"
not really
Received response:"WHAT DOES THAT SUGGEST TO YOU?"
nothing
Received response:"I SEE."
bye
Session ended by remote party.
```

Server

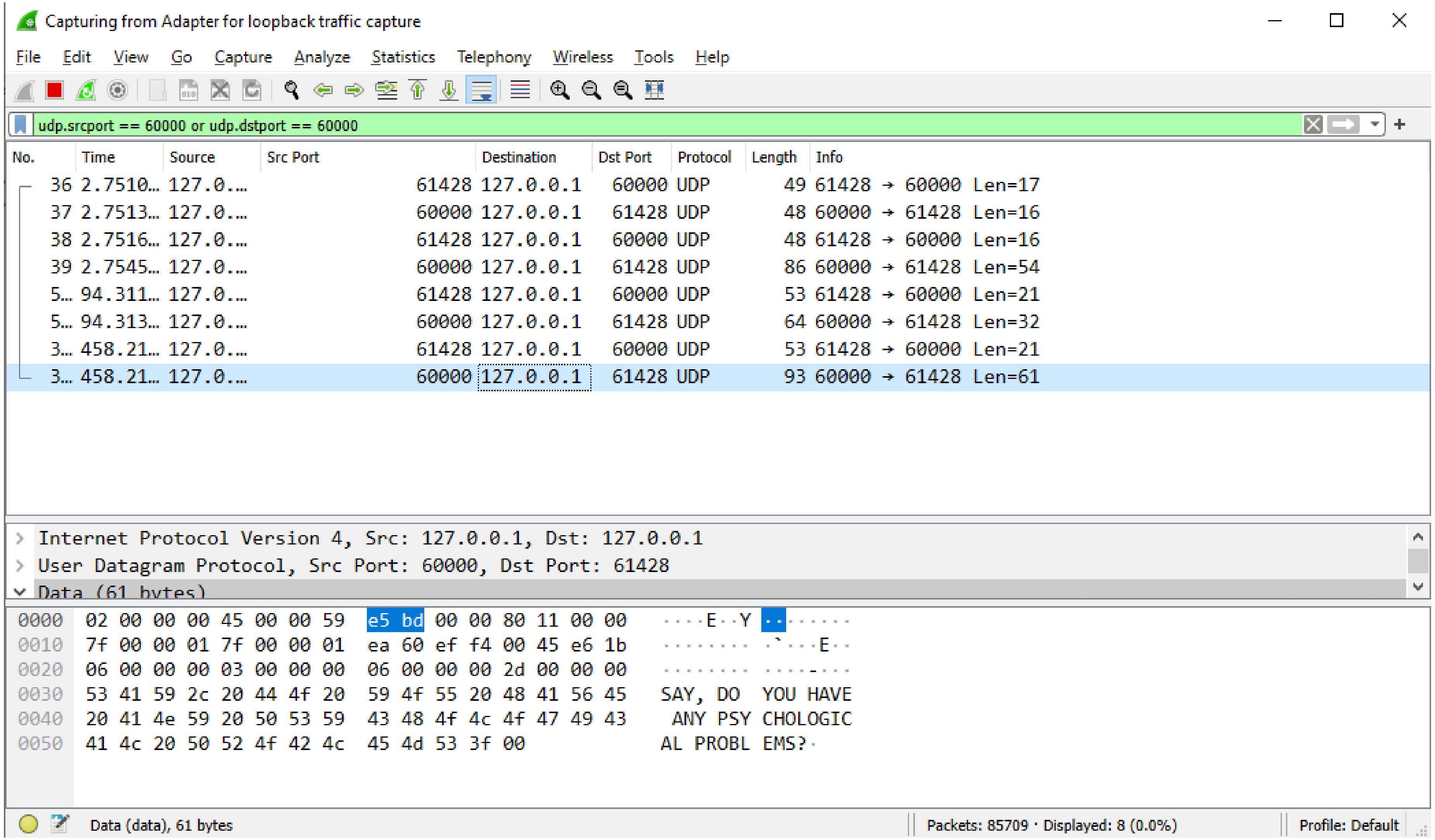
Command Prompt - udp_party -port 3000 -key alice.key -pwd alice -cert alice.crt -root rootCA.crt -peer Bob.com

```
C:\Users\alexber\OneDrive - Intel Corporation\Documents\GitHub\documentation.training.applied-crypto-part2\udp_party\x64
\Debug>udp_party -port 3000 -key alice.key -pwd alice -cert alice.crt -root rootCA.crt -peer Bob.com
Server: Starting listening...
New session 1 created with Bob.com
(1) Created
(1) Welcome: "HI!  I'M ELIZA.  WHAT'S YOUR PROBLEM?"
(1) Request: "I'm trying to write crypto code"
(1) Response: "DID YOU COME TO ME BECAUSE YOU ARE TRYING TO WRITE CRYPTO CODE?"
(1) Request: "yes"
(1) Response: "YOU SEEM QUITE POSITIVE."
(1) Request: "Writing crypto code is difficult"
(1) Response: "SAY, DO YOU HAVE ANY PSYCHOLOGICAL PROBLEMS?"
(1) Request: "anyone who writes crypto code has problems"
(1) Response: "WHY DO YOU ASK?"
(1) Request: "are you real?"
(1) Response: "WHY ARE YOU INTERESTED IN WHETHER OR NOT I AM REAL?"
(1) Request: "not really"
(1) Response: "WHAT DOES THAT SUGGEST TO YOU?"
(1) Request: "nothing"
(1) Response: "I SEE."
(1) Request: "bye"
(1) Closing.
```

Existing protocol details



Protocol capture



Options

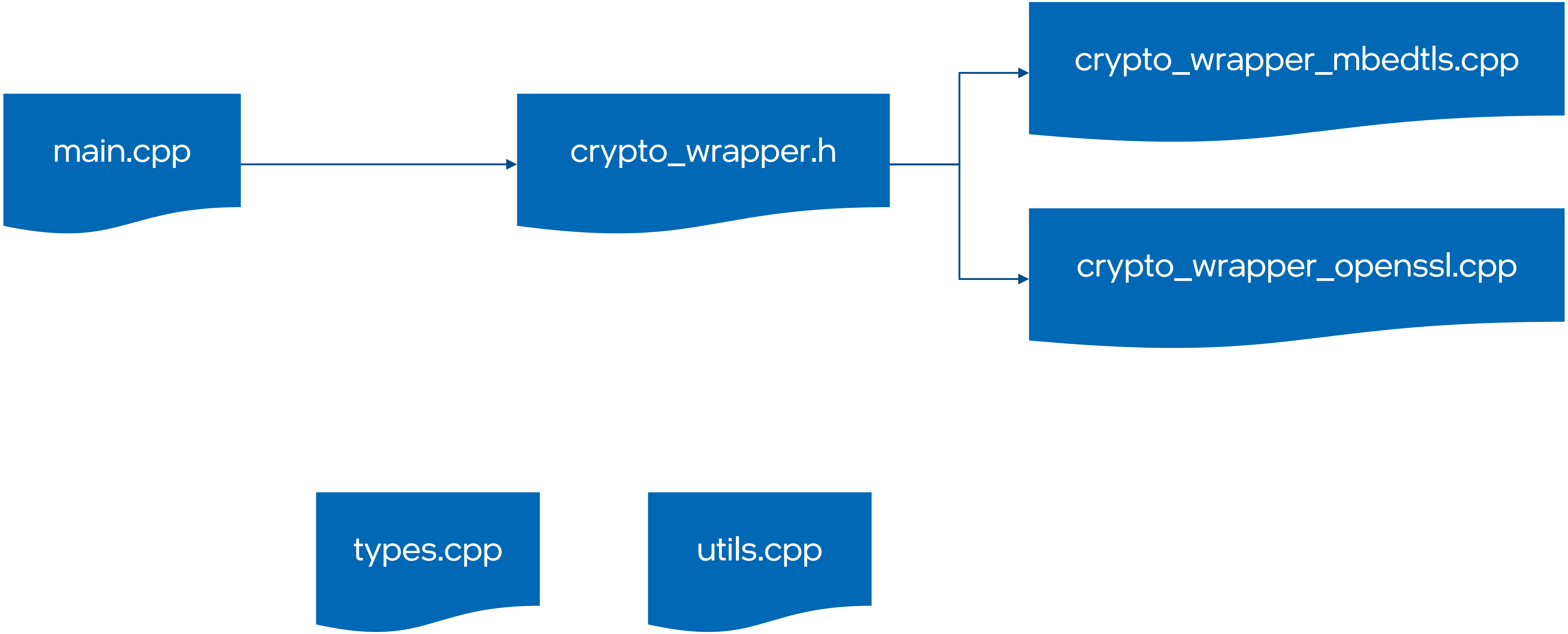
- Operating system
- Crypto library

OpenSSL
Cryptography and SSL/TLS Toolkit

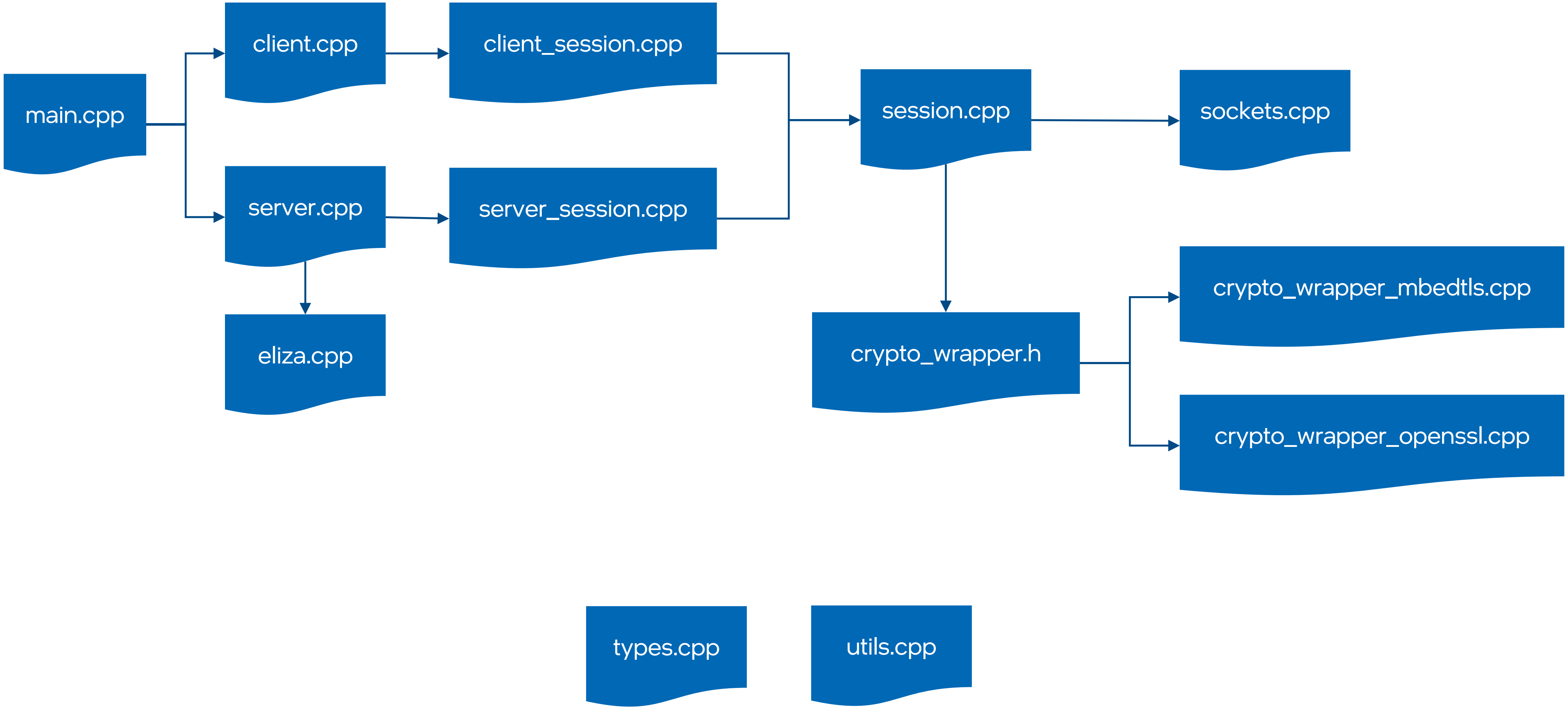
Mbed TLS

intel/ipp-crypto

Crypto Unit-test Code Structure

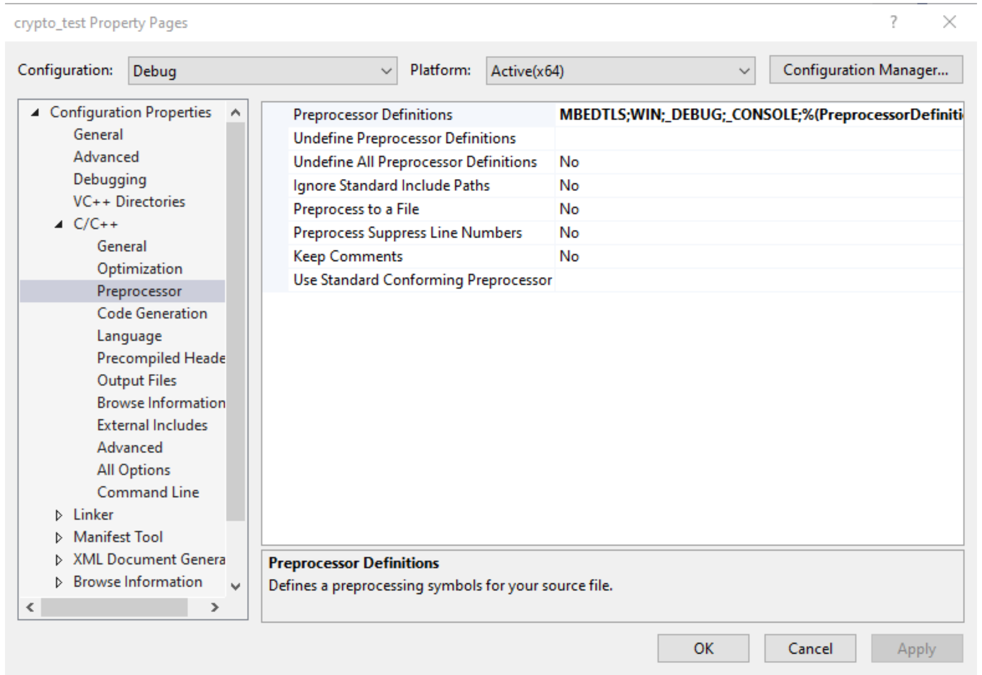


Code Structure



Building

- Windows
  - Use the udp_party.sln
  - Configure the MBEDTLS or OPENSSL Preprocessor Definition
  - Build the solution



- Linux
  - Modify the makefile
  - Run `make` or `make clean`

```
# uncomment the desired crypto lib line
#CRYPTO=mbedtls
CRYPTO=openssl
#CRYPTO=ipp_crypto
```

Running

Make sure you have the required key files and certificate files

For server

udp_party -port 60000 -key alice.key -pwd alice -cert alice.crt -root rootCA.crt -peer Bob.com

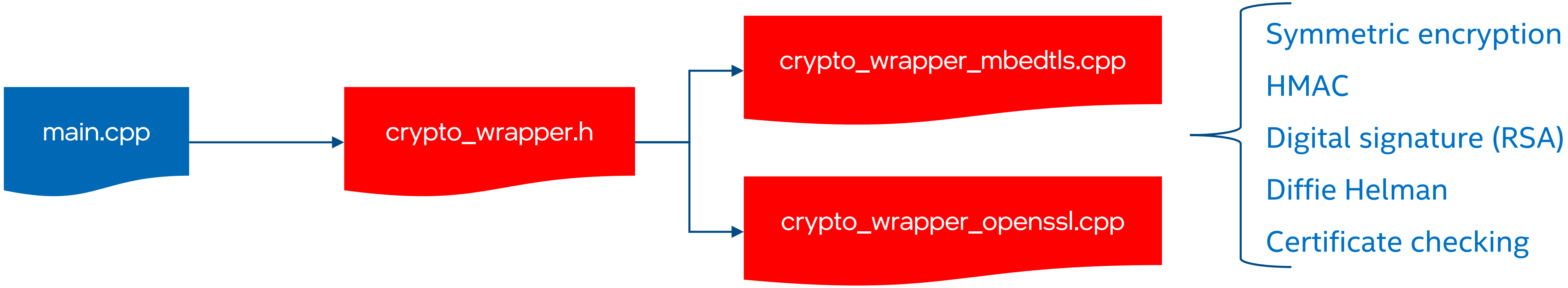
For client

udp_party -ip 127.0.0.1 -port 60000 -key bob.key -pwd bobkey -cert bob.crt -root rootCA.crt -peer Alice.com

Exercise – part I

- Implement crypto wrapper and make the crypto unit test pass

1. Get the code
2. Choose the crypto library to use
3. Install the crypto library
4. Configure your environment to the chosen crypto lib
5. Implement the missing crypto functionality in the chosen wrapper
6. Run the crypto_test project to see all tests pass



Crypto libraries details

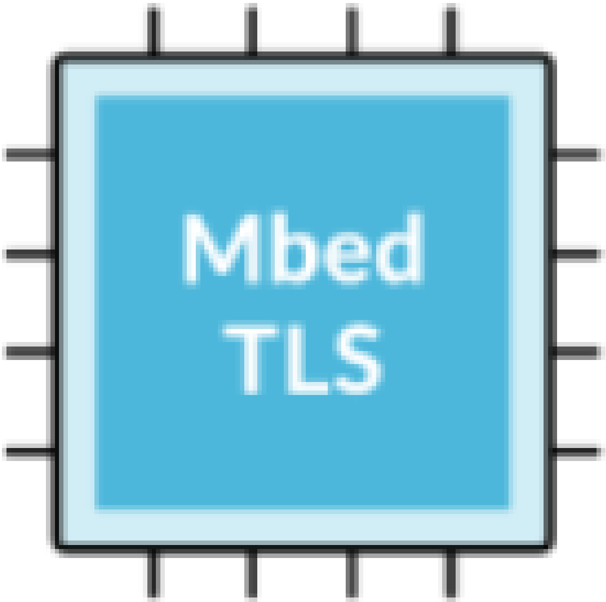
mbedTLS

- Download - https://tls.mbed.org/download
- Configuration – refer to https://github.com/ARMmbed/mbedtls
- Build:
  - Linux: make
  - Windows: open *mbedTLS.sln* file located in *|visualc|VS2010* folder
- udp_party config:
  - Linux – add *CRYPTO=mbedtls* to makefile
  - Windows add MBEDTLS to C/C++ Preprocessor Definitions project properties

OpenSSL

- Download, config and build:
  - Linux - https://nextgentips.com/2022/03/23/how-to-install-openssl-3-on-ubuntu-20-04/
  - Windows - https://slproweb.com/products/Win32OpenSSL.html
- udp_party config:
  - Linux – add *CRYPTO=openssl* to makefile
  - Windows add OPENSSL to C/C++ Preprocessor Definitions project properties

Useful APIs



- mbedtls_md_hmac
- mbedtls_hkdf
- mbedtls_gcm_setkey
- mbedtls_gcm_crypt_and_tag
- mbedtls_gcm_auth_decrypt
- mbedtls_md
- mbedtls_pk_get_type
- mbedtls_pk_rsa
- mbedtls_rsa_set_padding
- mbedtls_rsa_rsassa_pss_sign
- mbedtls_md_info_from_type
- mbedtls_rsa_rsassa_pss_verify
- mbedtls_dhm_set_group
- mbedtls_dhm_make_public
- mbedtls_dhm_read_public
- mbedtls_dhm_calc_secret
- mbedtls_x509_crt_verify

55

Useful APIs

OpenSSL
Cryptography and SSL/TLS Toolkit

EVP_MD_CTX_new
EVP_PKEY_new_raw_private_key
EVP_DigestSignInit
EVP_DigestSignUpdate
EVP_PKEY_CTX_new_id
EVP_PKEY_derive_init
EVP_PKEY_CTX_set_hkdf_md
EVP_PKEY_CTX_set1_hkdf_salt
EVP_PKEY_CTX_set1_hkdf_key
EVP_PKEY_derive
EVP_CIPHER_CTX_new
EVP_EncryptInit_ex
EVP_EncryptUpdate
EVP_EncryptFinal_ex
EVP_CIPHER_CTX_ctrl
EVP_DecryptInit_ex
EVP_DecryptUpdate
EVP_DecryptFinal_ex

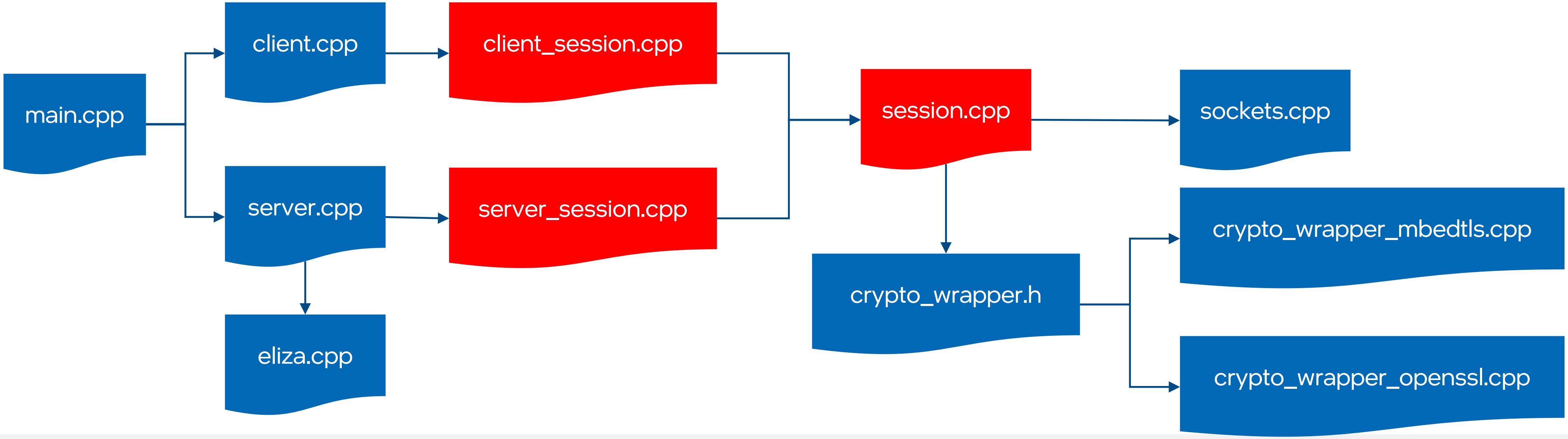
OSSL_PARAM_BLD_new
OSSL_PARAM_BLD_push_BN
EVP_PKEY_CTX_new_from_name
EVP_PKEY_fromdata_init
EVP_PKEY_fromdata
EVP_PKEY_CTX_new
EVP_PKEY_derive_init
EVP_PKEY_derive_set_peer
EVP_PKEY_derive_init
BIO_new
BIO_write
PEM_read_bio_X509
X509_STORE_new
X509_STORE_CTX_new
X509_STORE_add_cert
X509_verify_cert
X509_check_host

Useful resources

- https://github.com/Mbed-TLS/mbedtls (refer to Documentation section)
- Relevant mbedTLS files
  - hkdf.h
  - gcm.h
  - pk.h
  - rsa.h
  - entropy.h
  - dhm.h
  - bignum.h
  - md.h
  - x509.h
  - x509_crt.h
- https://cpp.hotexamples.com/
- https://github.com/openenclave/openenclave-mbedtls/blob/openenclave-mbedtls-2.16/programs/README.md
- Category:Examples - OpenSSLWiki

Exercise – part II (advanced)

- Add encryption and integrity protection over the traffic
- Use SIGMA for key exchange
  1. Enhance the session classes to use the crypto wrapper for SIGMA and channel protection
  2. Make sure the session is working correctly and securely

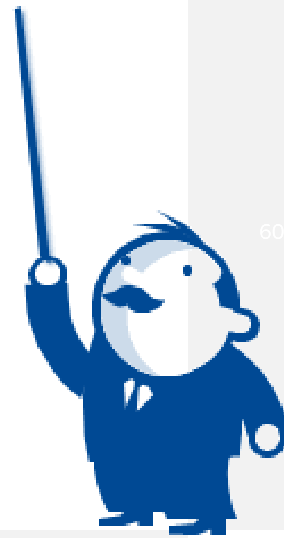


Key messages and resources

Summary

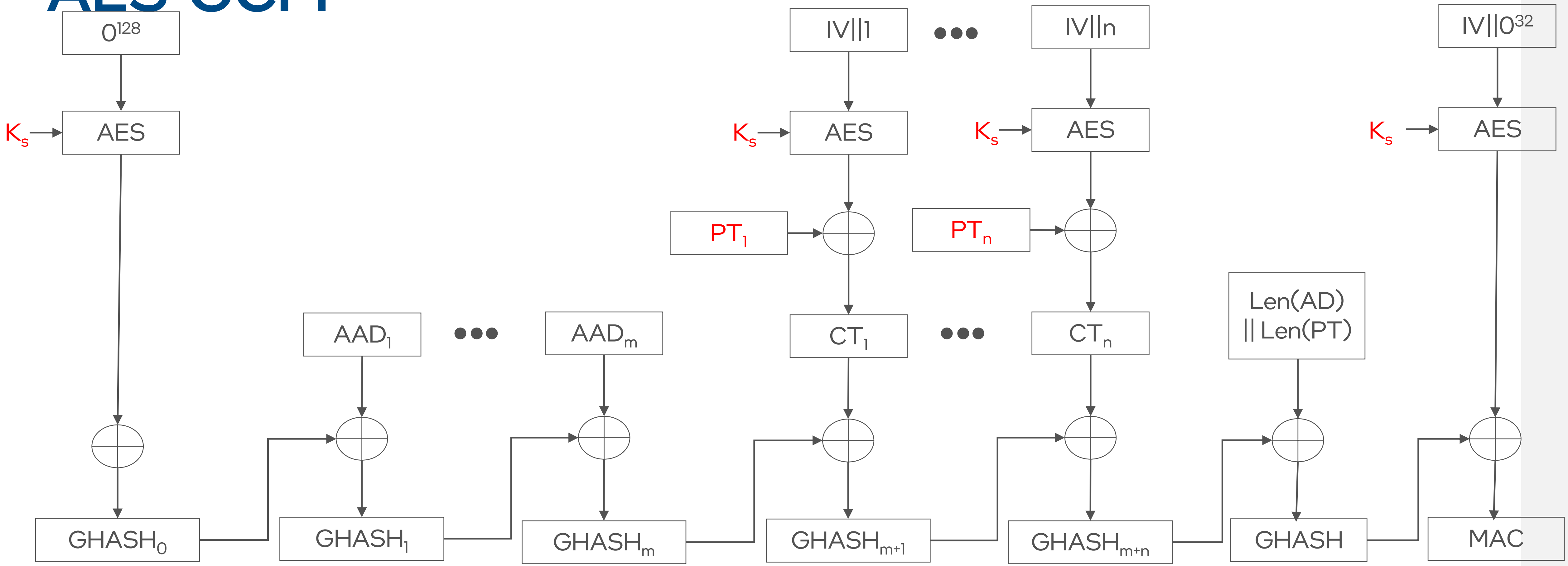
Course Objectives review

- Practice using recommended **cryptographic algorithms**
- Familiarize through practice with popular cryptographic libraries APIs
- Practice to avoid insecure **coding practices** with regards to using cryptography



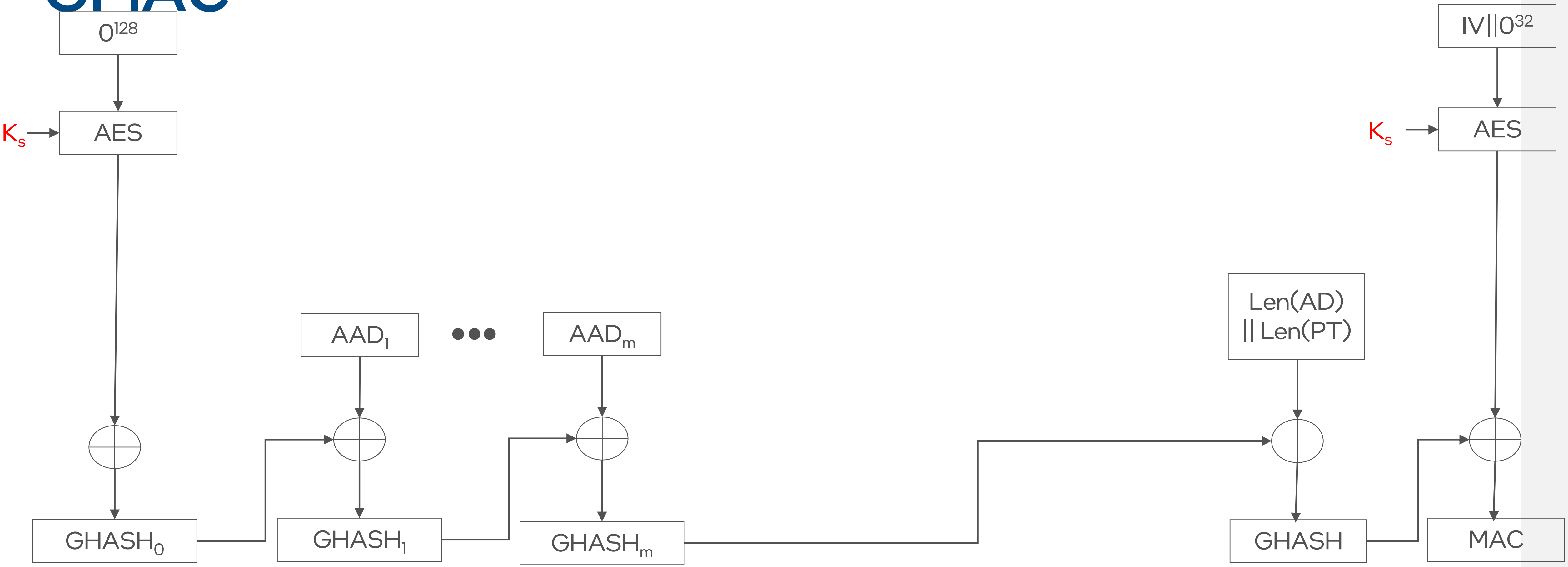
Backup

AES-GCM



K_s - secret AES key
PT_i – plaintext (block i)
CT_i – ciphertext (block i)
AAD_i – Additional Authentication Data (block i)
IV – Initialization Vector (96 bits)
0^i – i 0 bits
IV||i – IV (96 bits) concatenated with 32 bits representing i

GMAC



K_s - secret AES key
PT_i – plaintext (block i)
CT_i – ciphertext (block i)
AAD_i – Additional Authentication Data (block i)
IV – Initialization Vector (96 bits)
0^i – i 0 bits
IV| |i – IV (96 bits) concatenated with 32 bits representing i

- static constexpr unsigned int HELLO_SESSION_MESSAGE = 2;
- static constexpr unsigned int HELLO_BACK_SESSION_MESSAGE = 3;
- static constexpr unsigned int HELLO_DONE_SESSION_MESSAGE = 4;
- static constexpr unsigned int GOODBYE_SESSION_MESSAGE = 5;
- static constexpr unsigned int DATA_SESSION_MESSAGE = 6;

```
class MessageHeader
{
public:
    unsigned int sessionId;
    unsigned int messageCounter;
    unsigned int messageType;
    unsigned int payloadSize;
};
```