Secure Systems Design Exam

Academic Year: 2017/2018

December 21, 2017

Secure Messaging

using special-purpose devices

Realized By:

Francesco Parricelli

Aldo Strofaldi

Luca Pirozzi

Table of Contents

Introduction

Introduction

What is Secure Messaging?

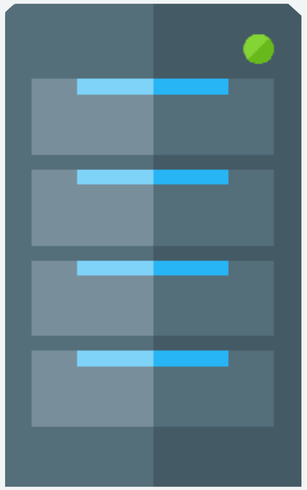
- Our idea was to made a *secure instant messaging application*.
- We used most of course's material (and other):
  - Secure Authentication;
  - Access Control Mechanism (based on role using XACML);
  - Encrypted Message Transmission (using custom protocol and SSL)
  - Special Purpose Prototypal Devices
  - And so-on...

Introduction

Entity Overview

End-User

SSL Connection

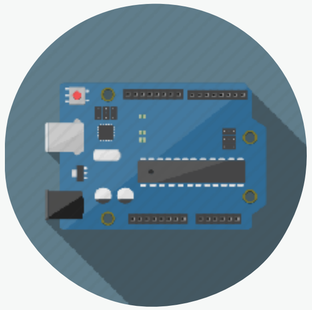


Application Server

SSL Connection

DataBase

Serial Channel



Application Overview

Application Overview

- The application we're presenting is intended to behave in an ICAM-like environment.
- ICAM (Identity, Credential and Access Management) is a comprehensive approach to managing and implementing digital identities (and related attributes), credentials and access control.

  It has been developed by the US government, but it is also applied in other environments (such as enterprise organizations and agencies).

Application Overview

- Four elements form the pillars of ICAM (although not all of them will be covered by our project):
- **Identity Management**
  - Concerned with defining a unique digital representation of an individual that can be leveraged across different internal departments and agencies for multiple purposes, such as access control.
- **Credential Management**
  - A credential is an object or data structure that authoritatively binds an identity to a token possessed by an individual (such as a smart-card, cryptographic keys and/or digital certificates): credential management focuses on the life-cycle of those credentials.

    In our context, we assume that a new employee of the organization must follow a registration procedure before being able to use the application: this process includes forging personal credentials for that user, usually in the form of a smart-card containg sensitive informations.

Application Overview

- **Access Management**
  - Defines the rules that govern the access to resources (both physical and logical) by the system entities: assures that the proper security controls are activated whenever an individual accesses a computer system, information data or physical buildings.
- **Identity Federation**
  - Manages how the individual's identity is treated when exported outside the organization or agency, and how an external individual's identity can be imported and managed inside the organization.

Application Overview

Private Certification Authority

- To best describe the infrastructure in more detail, we will focus on the registration steps needed by a new employee that wants to start using our application.
  - First of all, we assume that the organization owns a Private Certification Authority.
    In fact, since we want the users to be able to exchange messages in a secure way inside our organization, we must face the problem associated with the Public CAs: because they are public, those will issue a certificates for anybody, while our private service must insure only a selected group of people (or devices) have access.
    Therefore access to our private services need to be secured with certificates issued by our own private Certificate Authority.
    Many companies offer this kind of service, one of them being Symantec (link).
    In our case, we assume that this Private CA is in charge of providing (and signing) the digital informations that will form the user's credentials, for every user of the application (and ONLY for those users).

Application Overview

Private Certification Authority

- Therefore, our new employee enrolls for the credentials, a process which typically consists of identity proofing and the capture of biographic or biometric data.
- Once the necessary informations are collected, the credentials production process begins (involving the previously mentioned Private CA); at the end of this process, the produced credentials will be issued to the employee.

Application Overview

Private Certification Authority

- Those credentials are issued to the employee in the form of a physical device (smart-card) that contains the following sensitive informations:
  - The user's private key.
  - A digital certificate containing the user's public key, signed by our private CA.
  - The user's trust-store, containing the only identity that the user can trust: our private CA.
  - Those informations will be used to ensure security requirements while exchanging messages with other users and/or application servers.

Application Overview

Private Certification Authority

- Speaking of the trust-store, it contains the Private CA certificate: as previously said, every entity in the context of our application will have their certificate signed by our private CA.

  In this way, whenever the user wants to check the identity of an application-entity, he analyzes the provided certificate and looks into his trust-store to check if that certificate can be trusted: if the provided certificate is signed by the private CA (whose certificate is stored in user's trust-store) the user will be able to verify that the entity is actually a legitimate application-entity; otherwise, the communication will not start (we will rely on SSL/TLS so enforce those requirements).
- For realization and presentation purposes those informations are actually software-stored, but as we previously said in a real context they should be stored on a secure device with a strong protection policy (to avoid injecting untrusted identities into user's trust-store or stealing his private key).

Access Control

- The main actors in our application are:
  - Users, exchanging messages in a secure way using End-to-End Encryption (detailed later)
  - A Server, that the users will contact to perform authentication and authorization procedures (although those functions could be localized on different servers, they are presented as a single entity for realization purposes)

    This application Server will offer:
    - Authentication features (tied with MySQL database)
    - Authorization features
    - Certificate directory features
    - IDS-like features
    - Logging features
    - All the communications between client and servers will be properly secured by using SSL.

Access Control

XACML Policies

- Permissions for each role are defined through XACML Policies, enforced by our application Server.

  On the Server, a Java Servlet will handle all the contact-lists requests, permitting or denying them according to the XACML policy file.
- This Servlet:
  - Following the OWASP directives, uses a parameter-remapping function that prevents a malicious user to guess the correct POST parameters values to be sent to the Server
  - Controls that the user is properly authenticated before proceeding
  - If so, checks the policy file to verify that the authenticated user (using his role) has the right to access the requested contact list;

Access Control

XACML Policies

- Follows a Fail-safe default behavior, always denying access in presence of errors
- If the policy evaluates to Permit, picks the requested Contact List from the file system and sends it to the client; Client will be also configured following OWASP directives about processing external XML files to avoid XXE attacks
- Enables logging of every event of interest (permit and deny decisions, errors, and so on) to realize accountability and tracing; those logs will be automatically (and securely) sent to an external service (detailed later)
- Prevents access to the Policy File, blocking every request
- Invalidates Session at the end of every request processing, following OWASP directives

SSL Connections

SSL Connections

- We shortly describe how SSL handshake works, and apply those concepts to our architecture.

1. The handshake begins when a client connects to a SSL/TLS enabled server, requesting a secure connection: it presents a list of supported cipher suites (ciphers and hash functions)
2. From this list, the server picks a cipher and hash functions that it also supports and notifies the client of the decision.

3. The identity checking begins.

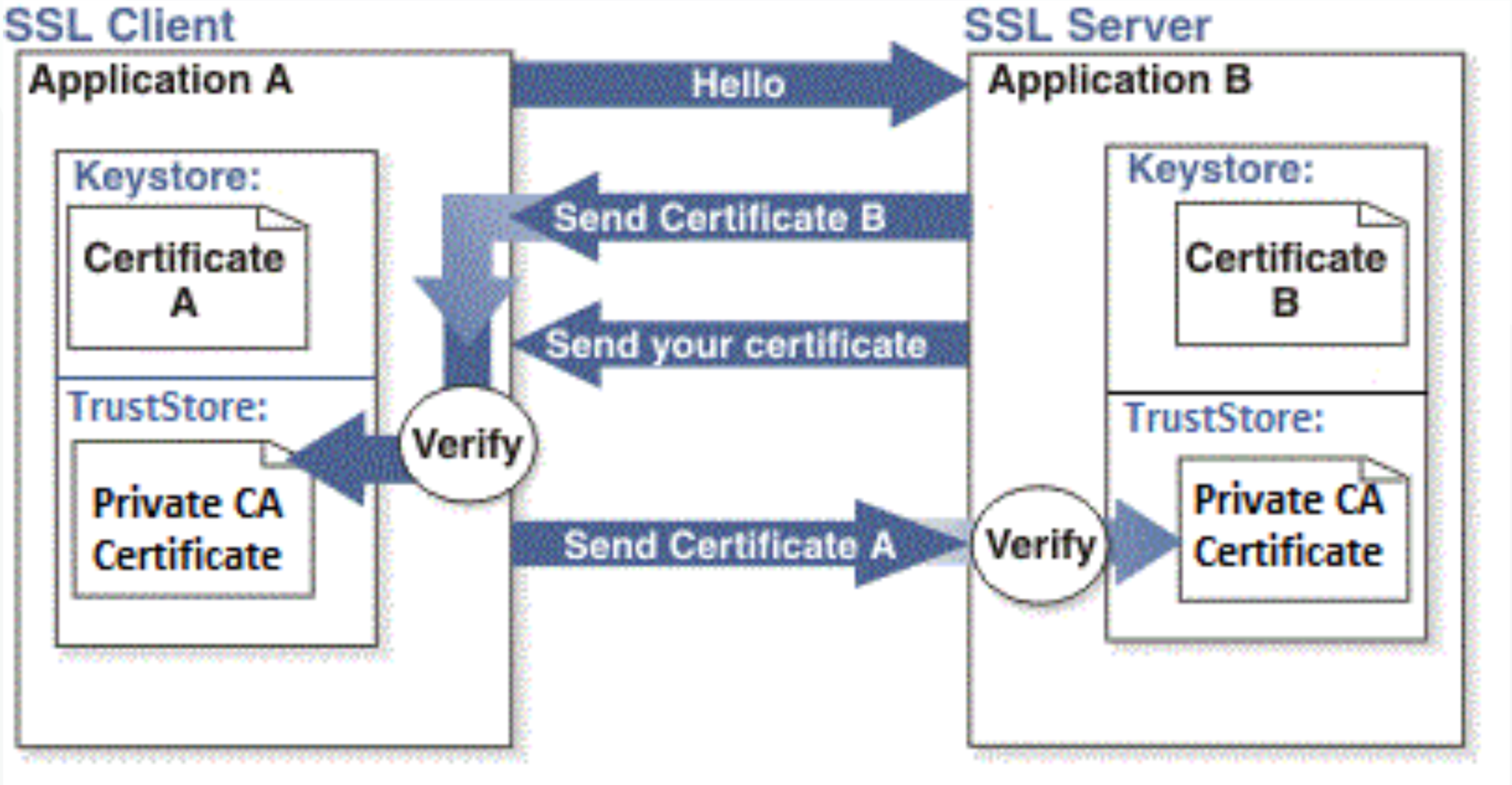
   In our case, we enabled 2 way SSL (SSL with client authentication).

   Server presents his certificate (signed by our Private CA) to the client, which looks into his trust-store (which only contains the Private CA's certificate) to check if the certificate can be trusted: having verified this certificate , the client now presents his own certificate (also signed by our Private CA) to the server, that performs the same check as the Client did before: since the Server *only trusts* our Private CA, it will verify the Client certificate and the process will go on.

SSL CONNECTIONS

4. Client can now generate the session keys used for the secure connection.
   This can be achieved by either encrypting a random number with the server's public key and sending the result to the server, by which both parties can generate a unique session key for the subsequent communications, or by using Diffle-Hellman key exchange to securely generate a random and unique session key.
5. When the session key is established, the secured connection can begin: messages will be encrypted and decrypted using that session key until the connection is closed.

SSL Connections



SSL Connections In Our Application

- Following OWASP guidelines, we enforced SSL connections all over the application.

  Being specific:

  1. We enforce SSL connection whenever a Client needs to communicate with the application Server, either for authentication or access control features.

     The application server is represented by a Tomcat server, and it has been configured securely by following the OWASP directives: disabling default/standard-applications, disabling Directory Listing, and enabling SSL with Client Authentication.

  2. We enforce an SSL connection whenever two users of the application wish to exchange messages: this is done by using Java SSL Sockets to setup the connection (in this case, the user starting the conversation will be the SSL Client, while the user waiting to be requested will play the part of an SSL Server).

SSL Connections

Benefits

- **Confidentiality**

  The handshake protocol defines a shared secret key that is used for symmetric encryption of SSL payloads.

- **Message Integrity**

  The handshake protocol also defines a shared secret key that is used to form a message authentication code (MAC).

  Secure hash functions (e.g., SHA, etc.) are used for MAC computations (HMAC).

- **Authentication (Client and Server)**

  As previously described, Client will check the Server identity against his trust-store, and the same operation will be performed by the Server; if Certificate verification fails, the handshake will be interrupted and the connection will not be established.

  Also, since the session-key setup is usually done (Client-side) using Server's public key, Certificate spoofing (entity pretending to impersonate our Server and redirecting communications) will be prevented, since the Server is the only owner of his private key.

Logging

Logging

Details

- To realize accountability, we enabled a logging service.
- This service is provided by LogEntries, which is part of the Rapid7 Solution Services (which also offers Metasploit Penetration Testing solution, covered by our previous presentation).
- It offers a complete compatibility with Java Source code, and can be configured on top of the popular log4j framework in a very easy and documented way.
- Whenever this service is enabled for an active (and registered) application, all the log4j API calls will be automatically redirected to LogEntries external service (in a secure way, using SSL)

Logging

Details

- Log entries generated by those calls will be available to analyze on LogEntries website, that offers a clear and simple GUI which can help to monitor events and accesses to our system (enabling, for example, to alert administrators on certain events), as well as to provide a form of non-repudiation against legitimate (or un-legitimate) user actions and related responsabilities
- By using an external service, we prevent those logs from being stored on the local file system (which may expose sensitive information, especially if they are intended to be reviewed by external individuals as OWASP directives say); we are also aware of the fact that for a complete log analysis we should use methods and techniques from the Log Analysis field (often implemented in many IDS).
- Also, an accurate consideration must be made about what type of information will be logged: in a real context, we must carefully choice the logging service provider, to be sure that all the collected information will be treated following certain parameters (confidentiality, in the first place).
- In our case, we decided to record the minimal information required (avoiding, for example, any potential stack trace): since the service we are using is a free service, this can be an acceptable compromise also if we think about the performance impact of an heavy logging activity on our system.

Application Server Features

Application Server Features No HardCoded Password

- Harcoding password in code can lead to failure of authentication measures and therefore access to critical resources, like databases
- We tried not to include hardcoded password in server code. For example, when accessing to a database or server mail account, we access to a file:
  - Containing username, password and other parameters encrypted in AES CBC mode, AES key and Initialization Vector
  - AES key has been encrypted using server's public key (so only who knows server's private key may read it)

Application Server Features No HardCoded Password

- The only exception is due to the fact we did not use a smartcard for private key and cryptographic functions, therefore we hardcoded server keystore password:
  - We could have avoided it asking for keystore password at server startup, but it would have been a palliative (it surely wouldn't be safe in a real application!)

Application Server Features SQL Injection Prevention

- Injection is included by OWASP in its Top 10 Web Application Security Risk list
- Injections happens when user input is not filtered or validated
- Database queries are run using Prepared Statements, which use placeholder before data evaluation, so query code and parameters are not mixed.
- Should this countermeasure not work, there are some sensitive queries with «LIMIT 1» constraints, which means that even if some data can be disclosed, attacker should retrieve them once a time (and provide different inputs for every attack)

Password Management

Password Management

Introduction

- Password management is critical for system security
- Storing them in plain text or using weak unsalted hashes leads to account stealing, which may be the first step to sensitive data disclosure or backdoor installation
  - An example of weak hash is MD5 which should be used only for digests, because is a fast, memory-conserving algorithm, easily crackable by GPUs, FPGAs and ASICs
  - Flame, a malware, was spread exploiting MD5-signed Microsoft Certificates
- Allowing easily-guessable password like «1234» or «password» is a vulnerability independent from hashing technique, so a strong password policy should be enforced.
- A system should be resilient enough even to password stealing, as instance implementing a multifactor authentication.

PASSWORD MANAGEMENT

- We analyzed several password cryptography techniques, like using not only a salt, but a pepper too
  - A pepper is like salt but it is not stored alongside the password.
  - Even though may seem adding security to passwords, no algorithm supports them, which means we would had to develop our cryptography algorithm and would probably reduce rather than enhance system security
- Among all hashing algorithms, we whittled it down to the ones designed to passwords: bcrypt, scrypt and PBDKF2.
  - New ones are being developed, like Argon2, but security administrator should not use too recent algorithms because there could be unknown critcial weaknesses in the system

Password Management

Comparising Table

- Estimated cost of hardware to crack a password in 1 year

KDF	6 letters	8 letters	8 chars	10 chars	40-char text	80-char text
DES CRYPT	< $1	< $1	< $1	< $1	< $1	< $1
MD5	< $1	< $1	< $1	$1.1k	$1	$1.5T
MD5 CRYPT	< $1	< $1	$130	$1.1M	$1.4k	$1.5 × 10^{15}
PBKDF2 (100 ms)	< $1	< $1	$18k	$160M	$200k	$2.2 × 10^{17}
bcrypt (95 ms)	< $1	$4	$130k	$1.2B	$1.5M	$48B
scrypt (64 ms)	< $1	$150	$4.8M	$43B	$52M	$6 × 10^{19}
PBKDF2 (5.0 s)	< $1	$29	$920k	$8.3B	$10M	$11 × 10^{18}
bcrypt (3.0 s)	< $1	$130	$4.3M	$39B	$47M	$1.5T
scrypt (3.8 s)	$900	$610k	$19B	$175T	$210B	$2.3 × 10^{23}

Password Management

Design choices

- We decided to use *bcrypt* because it works using a table thats is modified after every access, which makes GPUs ineffective in cracking it (unlike PBKDF2)
- While *scrypt* should provide better security, it is a relatively recent algorithm (2009), meaning that has not received enough scrutiny. Most recent feedbacks have downsized its reputation and its encryption is by far more computationally intensive. Moreover scrypt is designed for password based hard disk encryption, not password encryption.

Password Management

Design choices

- *bcrypt* needs more expensive hardware than PBDKF2 in order to crack it only if password is less than 55 characters long, which is bcrypt text length limit. Because a password is usually much shorter than 55 characters, that is not be an issue.
- *bcrypt* automatically adds salt to password, and it does not need to be stored; it encrypts passwords using key stretching.

Password Management

2-steps Authentication

- We implemented a 2-steps authentication system, so an attacker can't steal user identity by password knowledge only.
  - IP was used as identifier, but in a real-world scenario a more accurate device fingerprint would be needed (for example when server is used by private users)
  - It works under the assumption that the server is used by companies providing a static public IP: e.g. server is used by a company to provide secure communication and technical assistance for its employees (help desk)

Password Management

2-steps Authentication

- For every successful login attempt, device is compared to a list of trusted ones: if it's one of them, login request is approved; otherwise a confirmation mail is sent.
  - Those codes are not stored in plain text, but hashed with SHA-256.
  - Codes expire after 30 minutes. No new codes can be required for same (user,device) before expiration.
  - After issuing a new code, old one is deleted (attacker can't lead a replay attack with an old code)
  - If after less than 30 minutes right code is provided from that device, server adds it to the trusted devices list for that user.
- This way an attacker could steal an account only if:
  - Knows passwords and has one of the trusted devices
  - Or knows password and stole email account

AUTHENTICATION

AUTHENTICATION

- It was developed a token-based authentication system
- Firstly, client sends username and password to server
  - They are sent not encrypted, because HTTPS (SSL) will encode it (if we encoded it with same key, no security would have been added, just computation would have increased)
- If data are correct, server compares device data to a list of trusted devices for that account
  - If device is already trusted, user is authenticated
  - If not, a mail is sent to user's mail and user should provide also an activation code
- The server replies with a token
- We used JSON Web Tokens (JTW), an industry standard (RFC 7519)
  - Token is signed using server's private key (RSA)

Authentication

Token

- We have chosen to use stateless tokens
  - Server keeps no track of them
- A token contains:
  - Username
  - Device data
    - Protects against using token replays on untrusted devices
  - Issuing time (updated at every communication with server, see next point)
    - Protects against same token usage on stolen devices
  - Remaining uses (decreased at every iteration)
    - Protects against an attacker who manages to stole a device with an active session
    - Protects against spam

Authentication

Stateless Token

- The stateless feature has been added for a better scalability
- As server doesn't track issued token, there is no explicit token invalidation mechanism
- Including it would not require many changes to existing application
- It would need only a table that tracks user, device and logout time. When a new token is received for same user and device, time of issue is compared to last logout: if it happened before, token would have been invalidated.
- If user system crashed or user forgot client logged in, the usual token timeout would implicitly invalidate it.

Simple IPS

Simple IPS

- We developed a simple intrusion preventer with auditing and logging features
  - Its purpose is to stop unauthorized accesses to accounts, but it could be updated to include detection of suspicious activities from authenticated users
  - In a real application it should be combined with firewall
  - It is integrated in the application
- Keeps track of failed login attempts
- Can block specific account access for a specific IP
- Can block every access for a specific IP
- Such contermeasures are recommended by OWASP in order to reduce brute force/dictionary attack password efficiency and effectiveness

Simple IPS

Failed login attempts

- If a login attempt with existing user is failed, an entry (user,device,attempts,time) is generated
- On Client-side a captcha is generated after every failed login attempt.
- If the subsequently login is successful, the entry is removed in order not to block a legittimate user that simply mistyped the password
- If another login attempt with same user and ip is failed in a 15 minutes time span, the attempts counter is increased:
  - If more than 15 minutes have passed, a new entry is generaetd
- Maximum number of failed attempts is 5: if the attempts count reaches the maximum, a 20 minutes block on (user,device) is issued, and an account lockdown (user,device) entry is generated
- Every login attempt for that user and device in this 20 minutes interval will be rejected and the password not even be processed.
- These parameters have been generated randomly, but should be tailored to system workload or even be adaptive.

Simple IPS

Account and IP Lockdown

- System keeps track of number of account lockdowns issued for the same ip in an one day time span
  - Obviously, account lockdowns issued for the same user (and same ip) are cumulative
- If system issued an account lockdown to an IP for 5 times in less than a day, every login attempt from that IP will be ignored for 2 hours, without looking at user existence or password correctness
  - Another option could be blacklisting that IP.

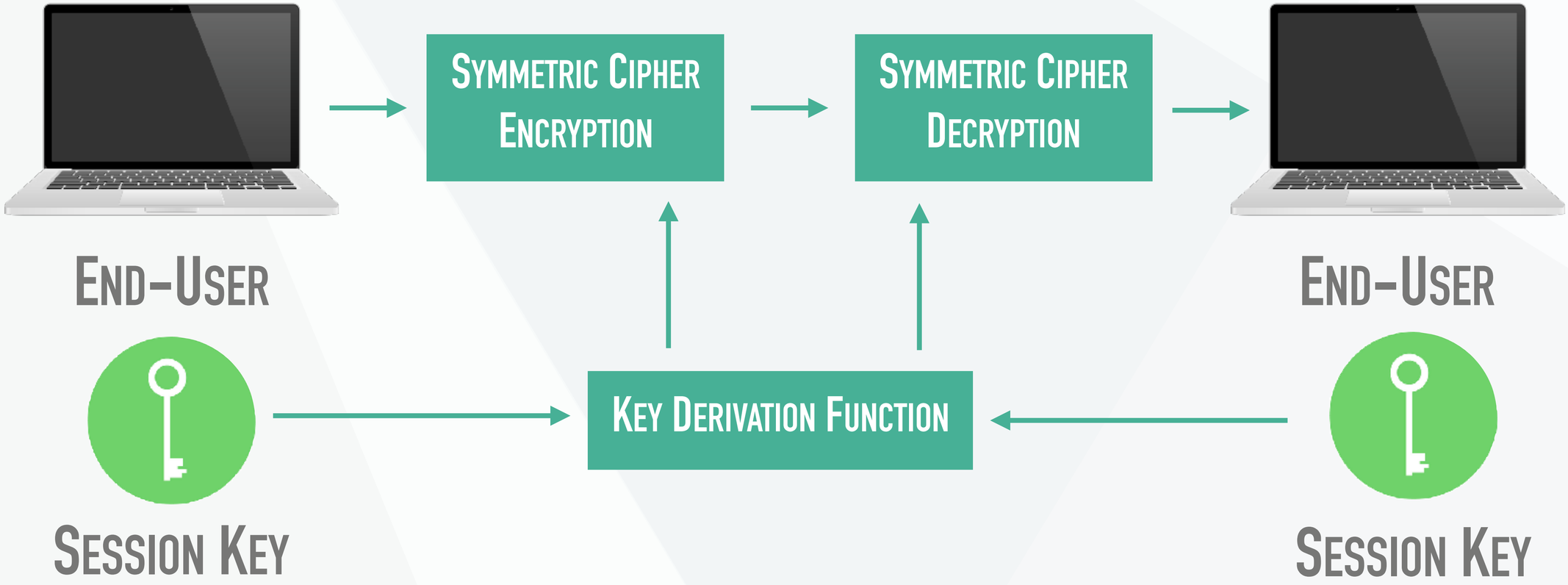
End-to-End Encyption

END-TO-END ENCRYPTION

- Most common instant messaging application introduced E2EE as a way to ensure that messages cannot be disclosed to unauthorized users.
- It is a way to realize a symmetric cryptography encryption, without explicit exchanging keys.
- It is adopted by various companies (i.e. Whatsapp, Telegram, Facebook Messenger)

End-to-End Encryption

How It Works?



End-to-End Encryption

Sharing Session Key

- Sharing *session key* is a problem!
- We can use asymmetric encryption (such as RSA) to do that....
- ...but it is slow to be executed on low-level devices.
- We could use a key agreement protocol: **Diffie-Hellman**
- In particular we adopted an Elliptic Curve Diffie-Hellman algorithm, *Curve 25519*.

End-to-End Encryption

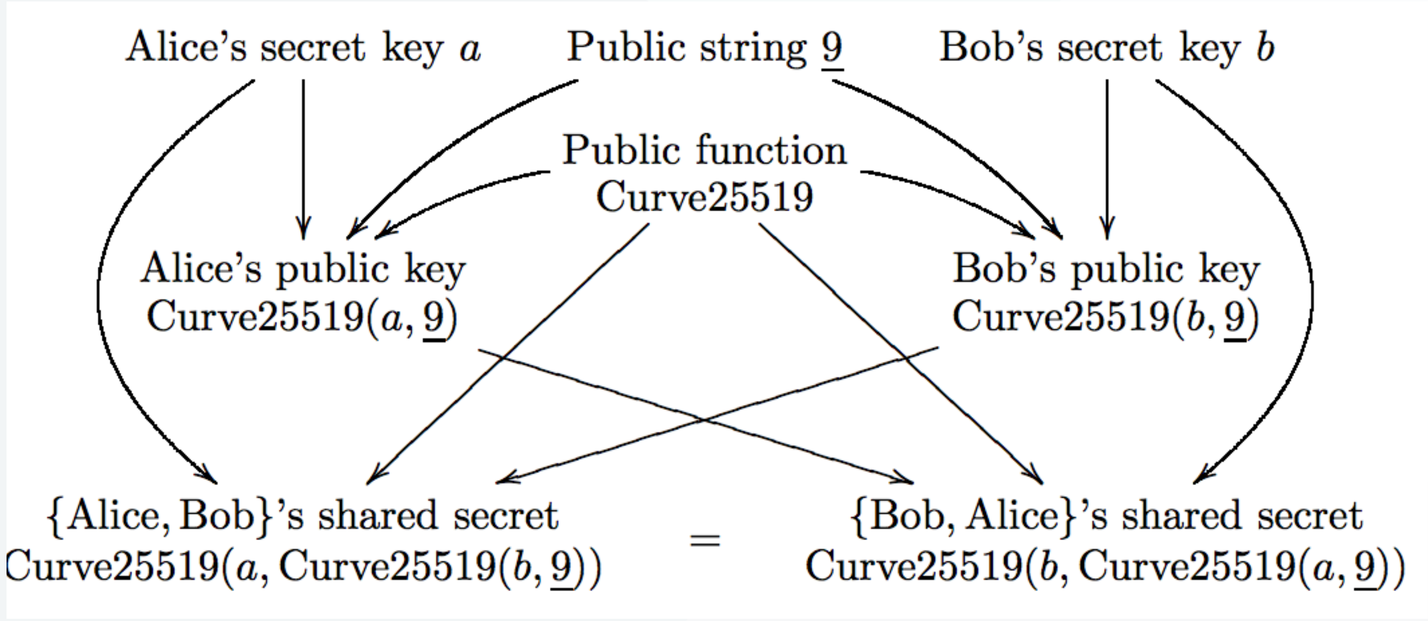
Curve25519

- Each Curve25519 user has a 32- byte secret key and a 32-byte public key.
- At the end of algorithm each set of two Curve25519 users has a 32-byte shared secret used to authenticate and encrypt messages between the two users.

End-to-End Encryption

Mid-Level View

- Because of shared secret key are equals, we can derive session key simply applying a Key-Derivation-Function



EFFICIENCY

- Extremely high speed (respect to other DH based algorithm).
- No time variability.
- Short secret keys.
- Free key validation.
- Short code.

End-to-End Encryption

- The legitimate users are assumed to generate independent uniform random secret keys.
- Let's choice a good Key Derivation Function can be broken (i.e. if a KDF returns a 64-bit string in which *k bits* are zero. This implies that a brute-force attack can easily break the session key).
- Replay attacks.

End-to-End Encryption

Implementation

- To gain E2EE to our application we based on Telegram MTProto Protocol.
- We customized it in order to have better performance on low-level devices.
- We used Arduino platforms to simulate an hardware token, in order to protect session keys.

End-to-End Encryption

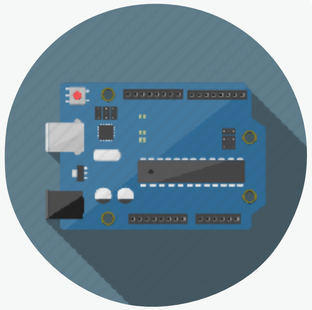
Handshake Overview

Send Public Token

Digital Signed

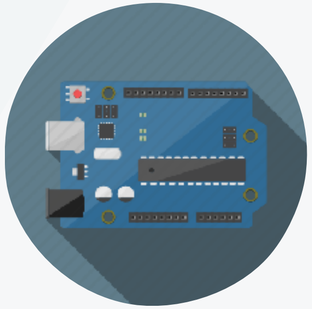
Each public shared Token has a Timestamp time-limited, that have to be checked to prevent Replay Attack

Calculate Secret and Public Token (randomly using a RNG)



Compute Secret Shared Key

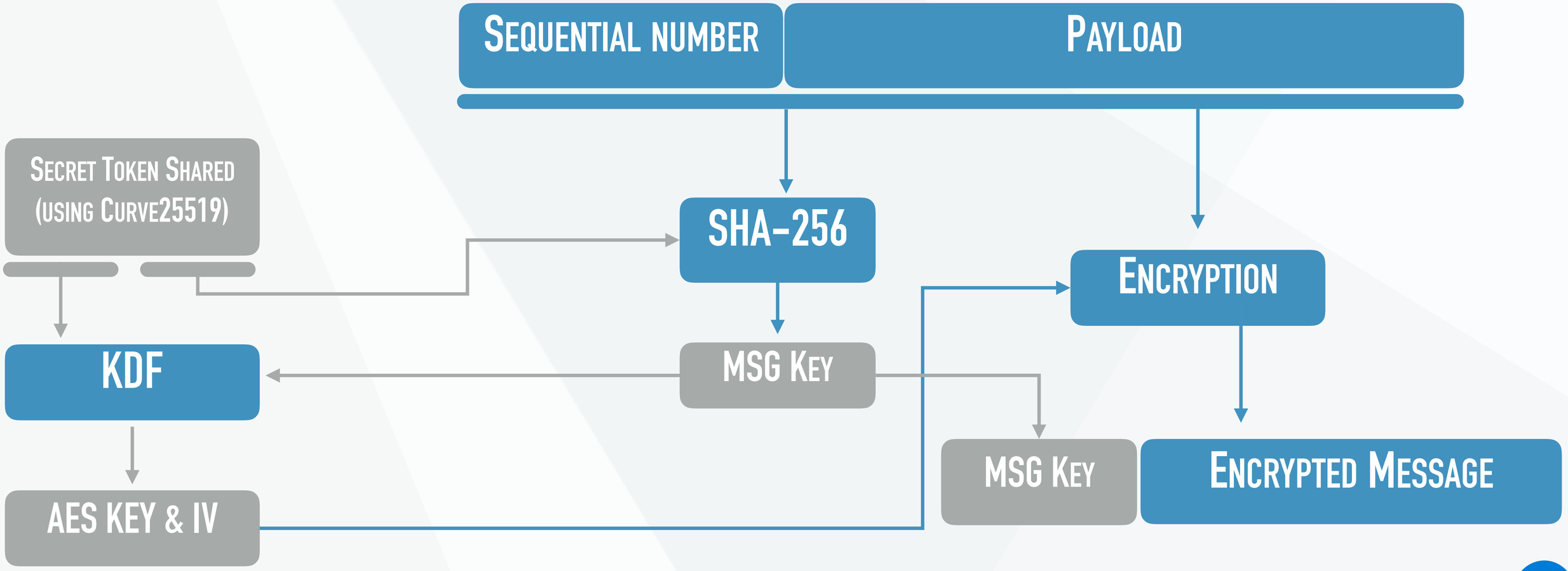
Compute Secret Shared Key



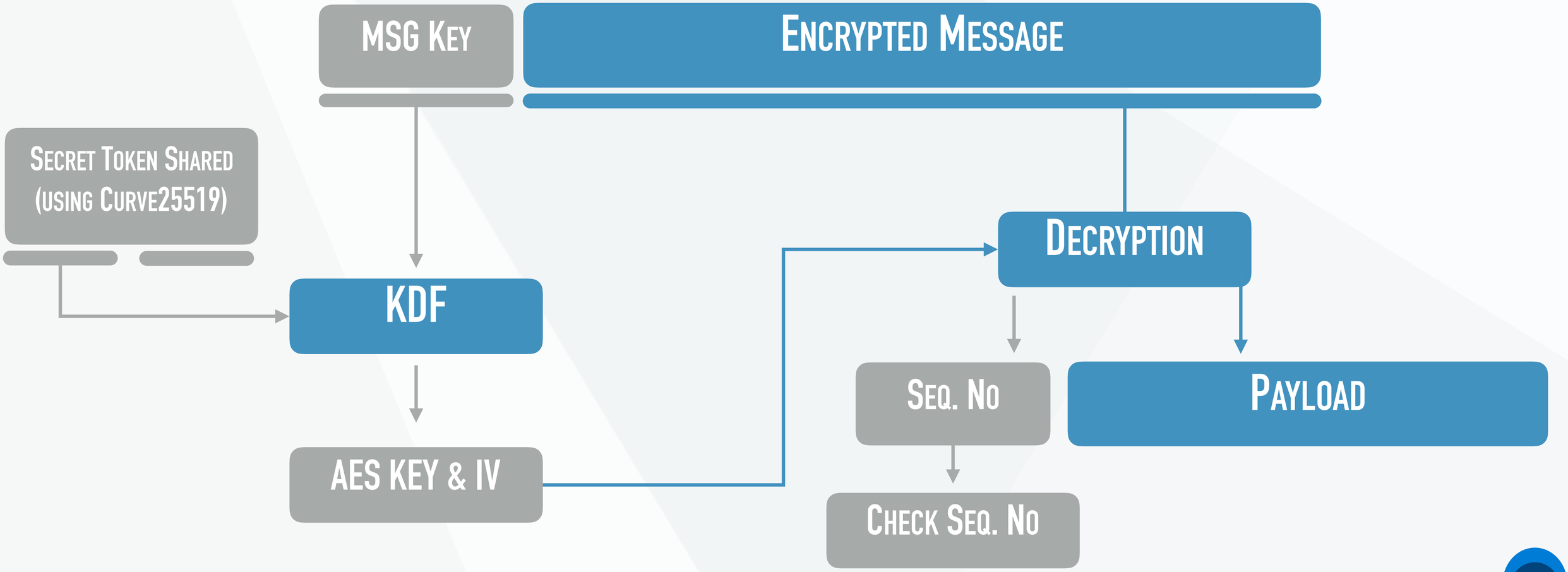
Calculate Secret and Public Token (randomly using a RNG)

End-to-End Encryption

Message Encrypted Sending



End-to-End Encryption Message Encrypted Receiving



End-to-End Encryption

HandShake Steps

1. The session starts when an end-point wants to begin a chat with another one.
2. End-point contacts its Arduino to get a random public key.
3. End-point use its own private key to digital sign the token and apply a timestamp in order to avoid replay attacks.
4. When the receiver receives the token, he contacts Arduino in order to get its public key.

End-to-End Encryption

HandShake Steps

1. The session starts when an end-point wants to begin a chat with another one.
2. End-point contacts its Arduino to get a random public key.
3. End-point use its own private key to digital sign the token and apply a timestamp in order to avoid replay attacks.
4. When the receiver receives the token, the application verify the digital signature obtaining trusted PK certificate from CA and check for the timestamp. Then he contacts Arduino in order to get its public key.

End-to-End Encryption

HandShake Steps

4. Receiver sends its digital signed and timestamped token to the other side, and call Arduino to perform the second phase of Curve25519.
5. Sender does the same.
6. Now both have got the same shared key that they use as session key.

End-to-End Encryption

Message Exchange

- Once the end-points defined a session, they can both communicate through a GUI console.
- For every message the application calls high level APIs to get encrypted message to send or to decrypt a message.
- The secret key never exit from Arduino.
- Arduino also traces about sequence number, in order to avoid replay attacks (this number is stored and changed in Arduino and it is part of *encrypted* message). To enforce the strength of this mechanism, Arduino handle a couple of sequence number, one used for transmission and one for received message and calculate the sequence number for current message considering who started session.

End-to-End Encryption

Encryption details

- As we sad before, once the session started, both end-points shares a secret session key.
- Session key is valid until the session is on.
- For every message a KDF calculates a new secret key using a part of session key and the message key, calculated as the SHA-256 computation of sequence number and message.
- The encryption is done by a CBC using AES-128 symmetric algorithm.

End-to-End Encryption

Overall Security

- If the RNG works well, the key exchange algorithm cannot be easily broken.
- Every known attack is more expensive than performing a brute-force search on a typical 128-bit secret-key cipher.
- Consider that if a brute-force attack on an encrypted message is successful then the attacker can only know about this message and not about the others.

Code Obfuscating

Code Obfuscating

- Anyone could download one of the thousand Java decompilers available on the Web, feed them with some bytecode, and retrieve (almost) the same originating source code.
- Possible solution: **byte code encryption.**
- Moreover, if we're going to decrypt an encrypted bytecode we also need a decryption key, and we must face the ever-lasting problem of key distribution and management.
- This works also in OpenJDK but…
- …it has been shown that OpenJDK can be easily modified to overcome any bytecode encryption.

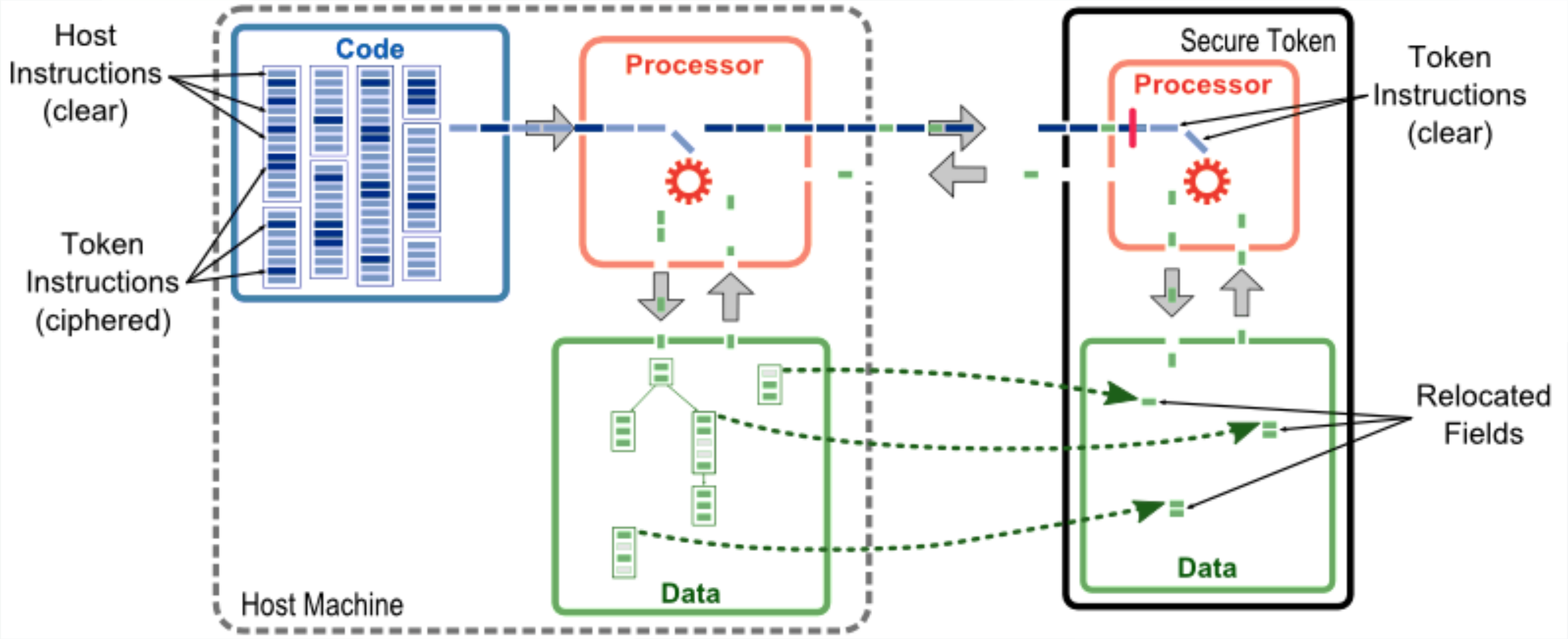
Code Obfuscating

Validy SoftNaos

- Validy SoftNaos seems a simple USB drive, but actually contains a slave co-processor that receives flows of encrypted instructions from the running application
- Interaction with a subset of the application state (also stored in token), subtracting a part of the application to run it inside the token.

Code Obfuscating

Validy SoftNaos: How works



Code Obfuscating

Validy SoftNaos example

- After standard Java compilation phase, the resulting bytecode is given to the SoftNaos Post-Processor, that translates the code of secure methods into the token instruction set, encrypts the result, and replaces the body of secure methods with sequence of calls transmitting the encrypted instructions to the token for execution.

```
import com.validy.technology.annotation.*;

public class Minimal {

        @SecureField
        private int counter = 0;

        @SecureMethod
        public void inc() {
                counter++;
        }
}
```

```
import com.validy.card.ValidyException;
import com.validy.technology.runtime.*;

public class Minimal
    implements Secured
{

    public Minimal(int i)
    {
        k$This = new Memory(i);
        Token.out(com.validy.technology.runtime.Secured.Pointer.get(this));
        Token.exe(0xce52da1a62f69204L);
        Token.exe(0x68d563976d888a64L);
        Token.exe(0x9aaa432570daf701L);
    }

    public Minimal()
    {
        this(1);
    }

    public void inc()
    {
        Token.exe(0x200e5d55a842110aL);
        Token.exe(0xdda147e8d0d556c1L);
        Token.exe(0x74256cd9420d9741L);
        Token.exe(0x6f53e373761060adL);
        Token.exe(0xde11625c9a743f29L);
    }
}
```

- On the right, we have the decompiled bytecode from the post-processed class files.

Code Obfuscating

Name Obfuscating

- The main goal of obfuscation is changing the code so that it produces the same results when run, but it is much more harder to understand and analyze when decompiled.
- It must be considered just as an additional layer of security, but not the *only* one
- *Name obfuscation* is the process of replacing standard naming identifiers (such as com.mycompany.security.checkPermission()) with meaningless sequences of characters, like a.a0().
- An interesting side effect of name obfuscation is the reduction of the class file size.
- Limitations: Standard Java API Classes and Serializable Classes names cannot be obfuscated

STRING OBFUSCATING

- *String encryption* is another feature commonly found in Java obfuscators.
- Replacing string literals with calls to a method that decrypts its parameter makes a hacker's life more interesting, but, unfortunately, not very much.
- The main problem is that those encrypted strings need to be decrypted at runtime, so the respective code must be included in the application.
- There are also more secure approaches like Stringer, which encrypts using encryption keys scattered across the obfuscated code

Code Obfuscating

Code Flow Obfuscating

- *Code Flow Obfuscation* deals with modifying a program so that it yields the same results when run, but is harder to decompile it into a structured form.
- Most obfuscators replace standard Java instructions with goto, in such a way that a decompiler cannot revert back the process (although not all decompilers are so easily defeated).
- Advanced Code Flow Obfuscation techniques include class hierarchy renaming, method inlining and outlining, loop unrolling, etc.
- Limitations:
  - Over obfuscated bytecode may not pass JVM bytecode verification
  - Code Flow Obfuscation has a negative impact on performance (for example, the previously mentioned Stringer has an impact on performance of about 10 % loss, as the official documentation reports)
  - Field Failure Data Analysis (FFDA) becomes more difficult (e.g: logging or stack traces)

Code Obfuscating Considerations

- Protecting Java bytecode is a very hard task, especially when this code is intended to be delivered outside our production environment.
- Although bytecode encryption may seem feasible, it comes with a strong layer of costs, complexity and implementation issues (as Validy SoftNaos shown).
- Bytecode obfuscation represents an alternative that could make the attacker's life harder, but does not provide a complete and assured protection *alone* (especially when moving towards open-source tools, we tested JShrink).
- However, several professional (and highly paid) services offer concrete solutions for this concern (Stringer, Proguard, Zelix Klassmaster)

Final Thoughts

Final Thoughts

Implicit weaknesses

- When developing an application, even if we strictly follow secure programming techniques, there are some residual potential weaknesses.
- We rely on programming languages
  - In recent Black Hat Europe 2017 Convention, Fernando Arnaboldi, Senior Security Consultant, has shown how securely developed application may have unindentified vulnerabilities in the underlying programming languages that may lead to remote code execution
    - Affected languages are common used ones like Javascript, PHP, Ruby, Perl and Python.

Final Thoughts

Implicit weaknesses

- Trusting third party-developed cryptography libraries means that:
  - A trapdoor would allow decryption without key knowledge
  - Library could contain intentional, laziness or mistake weaknesses, or backdoors, which could ease data decryption.
    - Detect such errors is not a trivial task.
- There could be flaws at algorithmic level
  - NSA has been blamed for Random Number Generation weaknesses in Dual_EC_DRBG algorithm, which was approved by NIST and used by important companies like RSA

Final Thoughts

Implicit weaknesses

- Even hardware can be exploited and firmwares should be regurarly updated
  - Modern CPUs have cryptographic APIs, that could have the flaws we discussed before.
  - Recent Intel CPUs had firmware faults for Management Engine, Trusted Execution Engine, Server Platform Services that could lead to tens of dangerous vulnerabilities, permit non-signed code execution that could not be discovered by CPU security measures or security softwares
  - There is a small Minix-running CPU in recent Intel CPUs, with ring -3 privileges (full privileges), containing :
    - Full network stack
    - File system
    - Many drivers
    - A web server (?)

FINAL THOUGHTS

- Management Engine fault we discussed before affected the Minix-in-Intel CPU, so remote code would have been executed with full privileges without software or hardware tracking.
- Google is working for removing Management Engine from its servers
- What if a malware is inserted at MINIX-CPU level so that every Intel-CPU generated cryptographic key is sent by MINIX webserver to an attacker?

Final Thoughts

- Configuring an ICAM-like system is not an easy task, especially when the users of the system need to act in a strictly secure environment, under the supervision of a Private CA.
- Strong security considerations must be made about:
  - User Registration (cross-checking provided identity with private investigations);
  - User Certification;
  - Initialization and Configuration of work tools (hardware/software) to enable a proper interaction with the other security infrastructures deployed in the system;
  - Life cycle management;
  - Security Training;
  - Infrastructure management;

Final Thoughts

Logging & Access Control Thoughts

- Access Control policies must be always kept up-to-date, reflecting changes that may happen in the organization's structure
- Together with the protection of *resources* (which is done by a proper configuration of the access control infrastructure), a security consideration must be made about the protection of *policies* themselves: what if an attacker gains access to the policy file and modifies it to overcome access control?
- To address this (and other) kind of problem, a systematic monitoring infrastructure must be enabled: tracking user activities inside the systems helps the administrators to detect anomalities and intrusions, as well as providing a useful proof in a potential legal dispute.
- By using logging, we can also overcome access control related brute force attacks: if an attacker repeatedly tries to access a resource for which he doesn't have the proper rights, we can detect this situation from real-time log analysis and activate an *honey-pot* like mechanism, by which the attacker is kept in a restricted-playground with fake resources enabling the background collection of informations useful for suing him.

FINAL THOUGHTS

- We use a strengthless version of current AES standard, AES-128 due to Arduino power.
- In our prototype application we assume that messages were cannot be long over 31 characters.
- Time to take handshake is too much.
- Base64 Encoding/Decoding over message exchanged between Arduino and Java Application.

References

References

- ICAM https://www.dhs.gov/safecom/icam-resources
- Symantec Private CA

  https://www.websecurity.symantec.com/security-topics/private-ssl
- OWASP parameter remapping https://www.owasp.org/index.php/Top_10_2013-A4-Insecure_Direct_Object_References
- OWASP Securing Tomcat

  https://www.owasp.org/index.php/Securing_tomcat
- OWASP XXE Attacks

  https://www.owasp.org/index.php/XML_External_Entity_(XXE)_Prevention_Cheat_Sheet
- Rapid7 LogEntries

  https://logentries.com/

References

- https://www.blackhat.com/docs/eu-17/materials/eu-17-Arnaboldi-Exposing-Hidden-Exploitable-Behaviors-In-Programming-Languages-Using-Differential-Fuzzing-wp.pdf
- https://www.tomshw.it/falle-firmware-intel-milioni-computer-aggiornare-89817
- https://www.networkworld.com/article/3236064/servers/minix-the-most-popular-os-in-the-world-thanks-to-intel.html
- https://schd.ws/hosted_files/osseu17/84/Replace%20UEFI%20with%20Linux.pdf
- https://www.owasp.org/index.php/Blocking_Brute_Force_Attacks
- https://www.owasp.org/index.php/SQL_Injection
- https://www.owasp.org/index.php/Use_of_hard-coded_password