**Enhancing security in Over The Air Updates: Challenges, Cryptography, and Solutions**

Alexander Cheung

22073032

UXCFPU-30-3

Project and Dissertation

Word Count: xxxx

Shape

Description automatically generated with medium confidence

# Abstract

* Short description of the project
* 1 or 2 paragraphs
* Summarises the work completed
* Usually written at the end of the project

# Acknowledgements

* Where you thank contributions to the project
* Written at the end of the project

# Table of Contents

[Abstract 1](#_Toc192468873)

[Acknowledgements 2](#_Toc192468874)

[Table of Contents 3](#_Toc192468875)

[Table of Figures 4](#_Toc192468876)

[Table of Tables 5](#_Toc192468877)

[Introduction 6](#_Toc192468878)

[Aims and Objectives 6](#_Toc192468879)

[Aims 6](#_Toc192468880)

[Objectives 6](#_Toc192468881)

[Literature review 8](#_Toc192468882)

[Introduction 8](#_Toc192468883)

[Challenges Of OTA Updates 8](#_Toc192468884)

[Compliance 8](#_Toc192468885)

[Security Solutions 8](#_Toc192468886)

[Conclusion 10](#_Toc192468887)

[Summary of work and review 10](#_Toc192468888)

[Solution 11](#_Toc192468889)

[Requirements 12](#_Toc192468890)

[Functional requirements 12](#_Toc192468891)

[Non-functional requirements 14](#_Toc192468892)

[Methodology 17](#_Toc192468893)

[Tools 18](#_Toc192468894)

[Sprints 19](#_Toc192468895)

[Design 19](#_Toc192468896)

[Implementation 19](#_Toc192468897)

[Testing 19](#_Toc192468898)

# Table of Figures

* Use Words table of figures feature

# Table of Tables

* Use Words cross-reference feature

# Introduction

Over The Air (OTA) updates are the process of distributing a new SoftWare (SW) or FirmWare (FW) packages to systems over wireless communications, such as Wi-Fi, cellular, and Bluetooth. Since being introduced and implemented as part of regular use from manufacturers, they have increased productivity, efficiency, and customer satisfaction through their seamless application and minimal, if any, impact on the customer. With a focus on the automotive industry, OTA updates have meant the reduced costs for engineers implementing updates on cars when they arrive at the garage, reduced the impact on customers time for them having to bring the vehicle to the garage, and the improved safety and security of the vehicle as updates can be downloaded and installed as soon as possible at safest time.

However, OTA updates come with their own challenges. These challenges include the low computational power of embedded systems resulting in a balance between security and efficiency needing to be found, the updates cannot be installed while the vehicle is in motion or operation, and current solutions lack high-enough grade cryptographic algorithms to be considered secure as well as lacking the proper validation and integrity checks. The challenges of OTA updates in the automotive industry are further detailed within the literature review.

The project outlined within this document investigates OTA updates security and the different methods that needs to be implemented to improve security whilst also accounting for the constraints of embedded systems and the application within the automotive industry. The project is completed on laptops which, whilst they have greater computing power than an embedded system, are there to simulate endpoint vehicles and the OTA communication method used will be Wi-Fi.

## Aims and Objectives

### Aims

The aims of the project describe the end goal and what is wanted to be achieved through the research.

1. To create a working two-way communication system between the server controlling the updates and the client receiving the updates
2. To implement best security practices, whilst accounting for embedded system constraints, into the created solution
3. To address the current challenges with OTA updates in the automotive industry, and provide a solution that solves these problems
4. To create the solution modularly with the thought of post-quantum algorithms implementation

### Objectives

The objectives of this project are the steps taken to achieve the aims of the project. The objectives will be measured for their success throughout the project.

1. The solution is to provide clear and simple communication across a network between the server and clients
2. The solution uses Wi-Fi as its primary communication to the clients
3. Efficient but secure cryptography will be implemented to address the computationally low power of embedded systems
4. Additional authentication, validation, and integrity checking will be implemented to further enhance security within the solution
5. Current algorithms and technology available will be used for security

# Literature review

## Introduction

The following literature review provides an introduction and overview to OTA updates which explores current and related research done by others on OTA updates, providing a background into current challenged of OTA updates, different standards that address OTA updates, current OTA update solutions, and different security methods for OTA updates.

## Challenges Of OTA Updates

OTA updates, whilst convenient and reduce costs for manufacturers, comes with an abundance of challenges. Highlighted by [Secure firmware Over-The-Air updates for IoT: Survey, challenges, and discussions (Saad El Jaouhari, Eric Bouvet)] are the most recognisable challenges when it comes to embedded systems and security:

* Low computational power
* Requirements for maintaining the security and privacy of the updates
* The discoverability and intractability of devices

Due to embedded systems not having an excess of resources for less important tasks, like security, it is often a challenge to implement and conduct security checks, like integrity and authenticity checks, alongside their primary functions. Further security and privacy requirements (as in GDPR) means that solutions are constrained to ensuring security it a priority, potentially impact core functionality. The embedded systems further need to be discoverable and have the capability to communicate so that they can relay important information, such as errors or core utilisation times (like when the vehicle is in motion) so updates are not performed during vehicle operation.

Other research from [Preparation and Deployment of Secure Over-The-Air Updates for Embedded Devices: Challenges and Solutions (Pavel Palurik, Michal Mikulasek, Lukas Jabloncik)] expanded upon the challenges to OTA updates in embedded systems:

* Reliability of updates
* Security risks
* Network coverage

### Compliance

[An adaptable security-by-design approach for ensuring a secure Over the Air (OTA) update in modern vehicles (Victormills Iyieke, Hesamaldin Jadidbonab, Abdur Rakib, Jeremy Bryans, Don Dhaliwal, Odysseas Kosmas)] research into the regulations, standards, and laws surrounding OTA updates, highlighting key artifacts that apply to OTA updates within the automotive industry. These include:

* ISO/SAE 21434 Road vehicles — Cybersecurity engineering: This is standard addresses requirements for the lifecycle of Electrical and Electronic (E/E) systems within road vehicles.
* ISO 24089 Road vehicles — Software update engineering: This standard addresses requirements and recommendations for software update packages
* ISO 20078-1 Road vehicles — Extended vehicle (ExVe) web services: This standard defines recommendations for implementations where an accessing party accesses web services
* UN R155 and UN R156

## Security Solutions

Various research projects have taken the idea of OTA upd

ates and proposed their own solutions.

[Secure firmware Over-The-Air updates for IoT: Survey, challenges, and discussions (Saad El Jaouhari, Eric Bouvet)] compares the two different modes for implementing OTA updates – Push and Pull. It also proposes the idea of a further hybrid mode which is a combination of the two.

|  |  |  |  |
| --- | --- | --- | --- |
| **Mode** | **Description** | **Pros.** | **Cons.** |
| Push | The server pushes updates to the client as and when updates are available |  |  |
| Pull | The client queries the server periodically for newly available updates |  |  |

[Preparation and Deployment of Secure Over-The-Air Updates for Embedded Devices: Challenges and Solutions (Pavel Palurik, Michal Mikulasek, Lukas Jabloncik)] proposes encrypting the update packages with a symmetric-key encryption algorithm to secure protect the update packages confidentiality. The research uses the Advanced Encryption Standard (AES) 128-bit (or, 256-bit) keys that are already assigned to the vehicles at time of manufacture, negating the need for transfer of sensitive key material. To verify integrity, the solution uses the Secure Hash Algorithm 256-bit version (SHA-256) to hash the encrypted update package which is then calculated again on the vehicles receiving end and compared with the received hash value. To validate the authenticity of the package, the Rivest-Shamir-Alderman 1024-bit (RSA-1024) digital signature algorithm is used which is then used to extract the hash from it and compared with the calculated hash.

[Attribute-based Access Control for Secure Firmware Over-The-Air Updates in Vehicles (Parth Shendkar, Yash Undre, Atharva Mahadik, Manas Borse, Rachana Yogesh Patil)] proposes a three-phase update process that uses attribute-based encryption to secure the update package. The specific attribute-based approach used is Ciphertext Policy - Attribute Based Encryption (CP-ABE) which uses an access policy to determine who can access and decrypt the data. The study showed the additional benefits to security that CP-ABE can add, as well as addressing the key challenges around OTA updates.

[Performance Evaluation of Attribute-Based Encryption in Automotive Embedded Platform for Secure Software Over-The-Air Update (Michele La Manna, Luigi Treccozzi, Pericle Perazzo, Sergio Saponara, Gianluca Dini)] is another study that researches the use of CP-ABE but focuses on its possibility to incorporate into current solutions. The key challenges of confidentiality and lack of encryption in current solutions is what the study aims to address. The study concludes that the performance impacts of CP-ABE on the update procedure are negligible and that it successfully increased security of current solutions with minimal impact.

[A secure and efficient vehicle OTA software update system: design and validation ()] proposes a hybrid encryption model which uses symmetric-key cryptography for encrypting the update package and then asymmetric-key cryptography for encrypting the symmetric-key. The study used AES for its symmetric-key algorithm, and RSA for its asymmetric-key algorithm. To validate the authenticity and integrity of the package, the SHA-3 hash within the RSA digital signature is extracted and compared against the hash generated on the receiving side. The study further conducts testing against the integrity and authenticity of the package as well as the time impact on the update process, concluding the increased security of the package and a reduced speed for the update process.

[Application and Research of Hybrid Encryption Algorithm in Vehicle FOTA System (Anyu Cheng, Jie Yin, Dongmei Ma, Xiaoyuan Dang)] also adopts a hybrid encryption solution where the symmetric-key is encrypted with an asymmetric-key. This study compares a range of symmetric-key and asymmetric-key algorithms for their speed, security, and resource consumption, concluding that for embedded systems, AES is best for symmetric-key cryptography and ECC is best for asymmetric-key cryptography. The research conducted focuses solely on the confidentiality and speed of the updates through analysis of the data package, showing that without the encryption the package can be read as plaintext, and through timings in the update procedure, comparing the times with and without encryption. The paper concluded that the AES-ECC hybrid encryption did improve the security of OTA updates and did not negatively impact the speed of the update.

## Summary

### Summary of work and review

|  |  |  |  |
| --- | --- | --- | --- |
| Work | Model | Pros | Cons |
| Preparation and Deployment of Secure Over-The-Air Updates for Embedded Devices: Challenges and Solutions (Pavel Palurik, Michal Mikulasek, Lukas Jabloncik) | * Uses AES for encrypting the SW image * Hashes the image using SHA-256 * Uses RSA-1024 for digital signatures * Verifies integrity and authenticity of the SW image |  | * Uses RSA-1024 which is computationally expensive and insecure due to low key length |
| Attribute-based Access Control for Secure Firmware Over-The-Air Updates in Vehicles (Parth Shendkar, Yash Undre, Atharva Mahadik, Manas Borse, Rachana Yogesh Patil) | * Three-phase model for implementing OTA updates * Uses CP-ABE for primary protection of the data |  |  |
| Performance Evaluation of Attribute-Based Encryption in Automotive Embedded Platform for Secure Software Over-The-Air Update (Michele La Manna, Luigi Treccozzi, Pericle Perazzo, Sergio Saponara, Gianluca Dini) | * Focuses on implementation of CP-ABE into current solutions * Assesses the impact of CP-ABE on the update process speed * Addresses challenges of confidentiality in current work |  | * Compares the time impact of CP-ABE to the other update processes, but a better comparison would be to other encryption methods |
| A secure and efficient vehicle OTA software update system: design and validation | * Uses AES for encrypting the SW image * Hashes the image using SHA-3 * Uses RSA for digital signatures * Uses RSA for encrypting the AES keys before they are sent OTA * Verifies integrity and authenticity of the SW image |  | * Uses RSA which, whilst a high key length can be used to overcome insecurities of the algorithm, is computationally expensive and not suitable for embedded systems * Transfers the keys OTA which is non-standard practice for automotive keys |
| Application and Research of Hybrid Encryption Algorithm in Vehicle FOTA System (Anyu Cheng, Jie Yin, Dongmei Ma, Xiaoyuan Dang) | * Concludes the best symmetric-key (AES) and asymmetric-key (ECC) algorithms * Uses AES for encrypting the SW image * Uses ECC for encrypting the AES keys before they are sent OTA |  | * Transfers the keys OTA which is non-standard practice for automotive keys * Only researches the use of ECC and AES in the update process and does not include any integrity and authenticity checks in their model |
|  |  |  |  |

### Solution

* Use proper simulated approach for keys being pre-loaded onto vehicles at time of manufacture (not key distribution OTA)
* No solution mentions secure connection between server and client
* Use AES -128 for symmetric-key encryption
  + Benefits from hardware acceleration
* Use SHA-256 for hashing
* Use ECC-384-DSA for digital signatures for efficiency
* Verify hashes of different stages of update process
  + After encryption and then data package before it is distributed
* Identify key challenges solved with this approach

# Requirements

## Functional requirements

|  |  |  |  |
| --- | --- | --- | --- |
| Req. No. | MoSCoW | Requirement | Explanation |
| FR-01 | MUST | The server and client MUST establish a connection | The server and the client must be able to establish a connection to each other to allow for communication operations |
| FR-02 | MUST | The symmetric keys MUST be retrieved from a secure location | The symmetric keys will be pre-loaded onto the devices (as they would at manufacturing) and must be pulled from a secure database on each device |
| FR-03 | MUST | The asymmetric keys MUST be retrieved from a secure location | The asymmetric keys will be pre-loaded onto the devices (as they would at manufacturing) and must be pulled from a secure database on each device |
| FR-04 | MUST | The server MUST be able to transfer and receive the data to and from the client | The server must be able to transfer and receive data to the client over the connection, including SW images, status codes, and error logs |
| FR-05 | MUST | The client MUST be able to transfer and receive the data to and from the server | The client must be able to receive data from the server over the connection, including SW images, status codes, and error logs |
| FR-06 | MUST | The SW image MUST be able to be encrypted | The SW image must be encrypted before has any operations executed on it or while it is in transit |
| FR-07 | MUST | The SW image MUST be able to be decrypted | The SW image must be able to be decrypted for use on the client side |
| FR-08 | MUST | The system MUST close the connection open | The system must close the connection between server and client to prevent unnecessary communication power being used |
| FR-09 | SHOULD | The SW image SHOULD be encrypted using symmetric-key cryptography | The SW image should be encrypted using symmetric-key cryptography for secure and efficient encryption |
| FR-10 | SHOULD | The SW image SHOULD be decrypted using symmetric-key cryptography | The SW image should be decrypted using symmetric-key cryptography for efficiency |
| FR-11 | SHOULD | The system SHOULD be able to generate hashes | The system should be able to generate hashes at different stages of the update process from server side to client side to verify integrity of the data |
| FR-12 | SHOULD | The system SHOULD be able to generate digital signatures | The system should be able to generate digital signatures to verify the source and authenticity of the data |
| FR-13 | SHOULD | The system SHOULD be able to generate Hash-based Message Authentication Codes (HMACs) | The system should be able to generate HMACs to verify both integrity and authenticity of the data |
| FR-14 | SHOULD | The system SHOULD be able to verify hashes | The system should be able to verify that hashes generated and received match or don’t match |
| FR-15 | SHOULD | The system SHOULD be able to verify digital signatures | The system should be able to verify that the digital signature is legit and authentic |
| FR-16 | SHOULD | The system SHOULD be able to verify HMACs | The system should be able to verify that HMACs generated and received match or don’t match |
| FR-17 | SHOULD | The system SHOULD handle errors and failures | The system should be able to handle any errors within the update process such as integrity or authenticity failures, or any general errors |
| FR-18 | SHOULD | The system SHOULD have a dashboard | The system should have a dashboard visible to show update progress of clients |
| FR-19 | COULD | The system COULD have a user authentication for the dashboard | The system could have user authentication for the dashboard which would require usernames and passwords for access |

## Non-functional requirements

|  |  |  |  |
| --- | --- | --- | --- |
| Req. No. | MoSCoW | Requirement | Explanation |
| NFR-01 | MUST | The system MUST use AES for symmetric-key encryption | The system must use AES for its symmetric-key encryption due to it speed, security, and efficiency within embedded systems |
| NFR-02 | MUST | The system MUST use SHA for hashes | The system must use SHA for its hashing algorithm due to its security and strength against collisions |
| NFR-03 | MUST | The system MUST use ECC-DSA for digital signatures | The system must use ECC-DSA for its digital signature algorithm for its efficiency and low computational power demand |
| NFR-04 | MUST | The system MUST use TLS for the secure connection | The system must use TLS for its secure connection to prevent unauthorised parties tampering or viewing data transmitted |
| NFR-05 | SHOULD | The system SHOULD use AES with a 128-bit key | The system should use AES-128 for encrypting and decrypting the update packages due to its hardware acceleration capabilities |
| NFR-06 | SHOULD | The system SHOULD use SHA with a 256-bit output hash | The system should use SHA-256 for generating hashes as it is secure for current technology and not excessively computationally expensive |
| NFR-07 | SHOULD | The system SHOULD use ECC with a 384-bit key | The system should use ECC-384 as it offers as much protection as the equivalent RSA key length, but with a fraction of the computational power demand |
| NFR-08 | SHOULD | The system SHOULD use TLS 1.2 or higher | The system should use TLS 1.2 or higher to comply with standards and avoid the use of deprecated products (e.g., TLS 1.1 or SSL) |
| NFR-09 | SHOULD | The security features SHOULD not excessively impact the speed of the solution | The speed impact from security features on the update process should not be excessive and be kept to a minimum |
| NFR-10 | SHOULD | The client SHOULD send status codes for the update process | The client should send back status codes to the sever at different stages of the update process to show successes and errors |
| NFR-11 | SHOULD | The client SHOULD have a status at all times | The client should always unicast to the server its status (e.g., update in progress, update pending, up-to-date, and needs update) |
| NFR-12 | SHOULD | The system algorithms SHOULD be implemented in a modular fashion | The system should be programmed in such a way that the algorithms can be swapped out to use post-quantum algorithms for a future proof system |
| NFR-13 | COULD | The system COULD take and store logs | The system could take logs from devices and store them for future recall |
| NFR-14 | COULD | The system dashboard COULD have secure usernames and passwords for logins | The dashboard could use secure usernames and passwords for authentication to the system |
| NFR-15 | WILL NOT | The system WILL NOT use post-quantum algorithms | The system will not use post-quantum algorithms as they are out of scope for this research |

## Use Case Diagrams

### Dashboard

# Methodology

* Agile methodology used
  + What is Agile?
  + Why has it been used?

# Tools

* Git/GitHub
* Jira
* VS Code
* OpenSSL/WolfSSL

# Sprints

* 2-week sprints

## Design

* System architecture
* Network-based diagram for low-level components
* Dashboard design
* Database design
* Flowchart
* Modular code design
* Testing process

## Sprint 1

A screenshot of a computer

AI-generated content may be incorrect.

* Devices all on the same network
* Clients wireless
* Server wired
* Management desktop used for remote connections to all devices due to limited hardware with monitors, keyboard, cables, etc. – would not be used in real life
* Client 1 – windows 10
* Client 2 – ubuntu desktop
* Server 1 – ubuntu desktop
  + Software will run on any os theoretically – ubuntu desktop was readily available and free

A screenshot of a computer

AI-generated content may be incorrect.

Devices on the network

A screen shot of a computer

AI-generated content may be incorrect.

Coordinator config

A computer screen with white text

AI-generated content may be incorrect.

Ping from server to clients

A screenshot of a computer program

AI-generated content may be incorrect.

Ping from linux client to server

A computer screen with white text

AI-generated content may be incorrect.

Ping from windows client to server

A screenshot of a computer

AI-generated content may be incorrect.

Github repository created, and different branches used

Ticket SCRUM-13 being worked on to create a simple backbone application for basic sending of data before any security or database storage is implemented

* Working for server and client agents
* Server always listens
* Clients are the ones that initiate the connections
* Using sockets

Simulation to show regular implementation – no security

Current workings:

* Build scripts
  + There is a build script to build the simulated agents into executables
  + There is a run script to run both agents
* Client
  + Can communicate to the server to check if there are any updates available
    - Returns True or False
  + It can download an update from the server
    - It asks the server if there is an update
    - It checks if it can have an update downloaded
      * This functionality will be applied to the installation function instead
  + It can change its update readiness status
  + It can display its current update readiness status
  + It can display the current update file
* Server
  + It pushes the latest update added to the database to the client
    - It first checks if the client can take the update and then pushes it
      * This functionality will probably be removed as not applicable to just downloading
* Both agents use threads
  + 1 thread for the options menu
    - Redisplays the options menu after every operation
    - For purposes of the simulation to allow for operations to be called at will
  + 1 thread for constantly listening for connections
  + 1 thread for waiting and servicing any active connections
* Uses simple sqlite3 databases to for information storage
  + This is some kind of secure hardware module in application
  + Database and storage technology is out of scope

Remaining working for the backbone application:

* Must
  + Server
    - Get the clients update readiness status
    - Get the clients current update status
      * + In application, the server should periodically check the client status if its not come back as installed to keep tabs on its status
      * Behind on update version
        + If version is same as newest, then it means it is installed
      * Downloaded
      * Installed
      * Value needs to be stored in server database
    - Get the clients current update file
      * Includes the version number
    - Add a new update file
      * Tell the client that there is a new update version
      * Pushes the update to the client
    - Before forcefully push and update, check the current update the client has and if it’s the same then don’t push the update
    - Accepts information relating to update version and adds it to the database
    - Accepts information relating to update readiness status and adds it to the database
    - Remove the need for checking if the client can download the update or not (update readiness status)
  + Client
    - The updates should be downloaded into the database to a new table and stored until they can be installed
      * Database acts as a buffer for the update before being installed
    - Remove checking update readiness status from downloading an update as that is for the installation of an update
    - It can take the update from the database and install it
      * Installs to another folder for simulation
    - Send update version to the server
    - Send update readiness status to the server
    - Send update installation status to the server
      * Checks the database for updates to be installed table and if its None then update is installed
    - Check if downloading a duplicate update
    - Change database update version and get rid of data file blob
    - If there is no download queued then the update was installed
    - Clear update queue after update is installed
    - Check update queue option in menu
  + Should use header files for file transfer
* Could
  + Logging
  + Web GUI for options