Finanziato dall'Unione europea NextGenerationEU
Ministero dell'Università e della Ricerca
Italiadomani PIANO NAZIONALE DI RIPRESA E RESILIENZA

Future generation Security for smart and connected Cars - FuSeCar

Deliverable D5.2: Transparent and secure vehicular communication protocols

WP5: Accountability and transparency for vehicular communications protocols and architectures

Authors:
Mattia Trabucco[2], Luca Ferretti[2], Giovanni Gambigliani Zoccoli[1], Mirco Marchetti[1], and Mauro Andreolini[2]
{name.surname}@unimore.it

[1]Department of Engineering "Enzo Ferrari"
[2]Department of Physics, Informatics and Mathematics
University of Modena and Reggio Emilia

Current revision: R1.1
Delivery date: February 20th, 2026

Revision history

Authors	Changes	Date	Revision
Mattia Trabucco, Giovanni Gambigliani Zoccoli, Luca Ferretti, Mirco Marchetti	Creation of the document, tentative structure, first draft of Section 2 and 3	November 28th, 2025	R0.1
Mattia Trabucco, Luca Ferretti	First draft of Section 2 and 3	December 15th, 2025	R0.2
Mattia Trabucco, Luca Ferretti	Draft of Section 4 and 5	January 12th, 2026	R0.3
Mattia Trabucco, Luca Ferretti	First draft of complete document	January 30th, 2026	R1.0
Mattia Trabucco, Giovanni Gambigliani Zoccoli	Updated document with new framework architecture, minor fixes	February 20th, 2026	R1.1

Finanziato dall'Unione europea NextGenerationEU
Ministero dell'Università e della Ricerca
Italiadomani PIANO NAZIONALE DI RIPRESA E RESILIENZA

Contents

1 Introduction

This deliverable is a technical report including the result of Work Package WP5: Accountability and transparency for vehicular communications protocols and architectures. This WP has one main outcome: propose a novel approach to support transparency in Over-the-Air (OTA) software updates for connected vehicles, ensuring the integrity and authenticity of updates while protecting the confidentiality of proprietary software.

The rapid evolution of Intelligent Transportation Systems and Vehicle-to-Everything communication has transformed the automotive industry, turning modern vehicles into highly complex, software-driven systems. Because software controls critical safety, navigation, and communication functions, the ability to deploy efficient and secure software updates is one of the most critical aspects of any modern information system. Updates are essential as they enable the addition of new security features, minimize the delay in patching disclosed vulnerabilities, and improve the overall security of the vehicle [1, 5]. Historically, software updates have relied on centralized distribution mechanisms, but these conventional designs present a lucrative and frequently attacked target. Securing the vehicular update infrastructure requires addressing several critical challenges:

- Single points of failure. The integrity and authenticity of updates traditionally depend on a single signing key, which is prone to loss or theft. Furthermore, relying on a centralized distribution infrastructure or a single trusted party creates a single point of failure: compromising it allows attackers to inject malicious code or subvert the update process;
- Confidentiality and access control. Unlike open-source projects, automotive software is highly proprietary. Proprietary software update systems must guarantee access control to prevent unauthorized clients from installing unauthorized updates, and ensure confidentiality to protect from reverse engineering and automatic exploit generation.
- Lack of transparency and timeliness. The current infrastructure for software distribution lacks transparency mechanisms, leaving room for stealthy backdooring of updates by compromised or malicious actors. Additionally, attackers executing man-in-the-middle attacks could mount replay or freeze attacks to prevent vehicles from receiving critical safety patches, making the timeliness of updates a vital security requirement.

This document outlines a comprehensive decentralized software-update framework [2, 3] specifically tailored for V2X nodes and automotive systems based on [7]. This framework is designed to be resilient against various attack vectors, including key compromise, insider threats, and supply chain attacks, while also ensuring that the update process is transparent and auditable by authorized parties.

2 Background

To establish a secure, survivable, and transparent software update framework for vehicular networks, this architecture relies on several advanced cryptographic and distributed system paradigms. This section outlines the core building blocks used to secure the update lifecycle.

2.1 Multi-Authority Ciphertext-Policy Attribute-Based Encryption

Attribute-Based Encryption (ABE) extends traditional public-key encryption paradigm by enabling fine-grained access control over encrypted data. Rather than encrypting to a single recipient public key, ABE ties ciphertexts and decryption keys to descriptive attributes and/or access policies, so that decryption is possible only when predefined conditions are satisfied. In this work, we focus on Ciphertext-Policy ABE (CP-ABE), where the encryptor embeds an access policy directly into the ciphertext at encryption time [6]. Each user private key is bound to a set of attributes describing the user's properties or roles, so that a ciphertext can be decrypted only if the attributes associated with the key satisfy the policy specified in the ciphertext.

While ABE provides powerful access control capabilities, it typically relies on a single trusted authority to generate decryption keys based on user attributes. To enhance the survivability of the framework, Multi-Authority CP-ABE (MA-CP-ABE) schemes allow multiple independent authorities to issue decryption keys, reducing the risk of a single point of failure. By carefully designing the access policies, we can tolerate the compromise of a threshold of key-generating authorities while ensuring that the confidentiality of both past and future updates remains intact. This means that even if some authorities are compromised, attackers cannot decrypt updates without satisfying the access policy, thus maintaining the security and integrity of the entire framework.

2.2 Multi-signatures and collective signing

Multi-signatures allow a specific subgroup of signers, or witnesses, to collectively sign a message, proving to a verifier that all subgroup members participated in producing the signature. Using protocols like CoSi [9], the system can efficiently produce compact multi-signatures that scale to large groups. This ensures that no single developer or administrator can unilaterally approve a software release.

2.3 Skipchains

While traditional update systems rely on a centralized server to provide the latest software version, a decentralized approach requires an immutable, publicly verifiable timeline of updates. Tracking a full blockchain (i.e., a distributed, append-only, tamper-evident distributed ledger used for logs) that stores public cryptographic material and software update metadata is often too resource-intensive for constrained devices like V2X nodes. Skipchain [8] is a novel authenticated data structures that combine the properties of blockchains and skiplists. A skipchain uses standard cryptographic hashes for backward links to past blocks, and also introduces forward links, created as digital signatures, that point to future blocks. This design enables resource-constrained clients to securely and efficiently validate binary updates without the need to continuously track a release chain by privately exchange, gossip, and independently validate on-demand newer or older blocks due to the skipchain forward and backward links being offline verifiable.

2.4 Reproducible builds

A critical vulnerability in software distribution is the inability to prove that a compiled binary perfectly matches its published source code, as compilation is often influenced by non-deterministic properties in the build environment. Reproducible builds are software development techniques that enable users to deterministically compile

a given source code into the exact same binary, independent of factors like system time or the specific build machine. By using reproducible builds, independent verifiers can compile the source code and cryptographically prove it matches the binary provided by the developers, protecting clients from compile-time backdoors and false security claims.

3 System and threat model

3.1 System model

The system models a proprietary software update scenario involving three primary entities: a software house, the distribution infrastructures, and the clients.

The software house encompasses several roles with shared interests. We denote as developers a set of n_d employees who access the source code, compile it, produce software binaries, and approve new versions. Each developer holds a signing key pair. Admins are a group of n_a administrators responsible for managing security-critical cryptographic material and defining the access control policies used for encrypting the binaries. Within the software house, we also have a set of n_r Attribute-Based Encryption (ABE) servers responsible for issuing decryption keys to authenticated clients based on specific attributes. Lastly, the software house maintains an authentication server that manages the database of registered clients and assigns them attributes upon successful authentication.

To distribute the software updates, the software house relies on an untrusted distribution infrastructure, such as a Content Delivery Network (CDN) or community-managed mirrors, which stores the encrypted software updates and their location information. The framework ensures confidentiality and enforces policy-based access control over this untrusted network using Multi-Authority Ciphertext-Policy Attribute-Based Encryption (MA-CP-ABE).

Finally, the clients, that represent the end devices (e.g., connected vehicles) that execute the software. A client periodically queries the infrastructure for new updates and maintains a set of ABE keys to decrypt the downloaded binaries.

The framework also relies on two additional roles to maintain the transparency and integrity of the update process: validators, a set of n_v entities that audit and validate new software updates through reproducible builds, holding a signed copy of the source code for this purpose, and witnesses, a set of n_w nodes that maintain a multi-layer skipchain using a Byzantine-fault-tolerant consensus algorithm. The skipchain serves as a public release log, ensuring that clients can efficiently validate updates and signing keys, guaranteeing transparency and freshness without relying on a centralized authority.

3.2 Threat model

The framework assumes a computationally bound attacker who cannot break the underlying cryptographic primitives. The attacker's goal may be to violate update confidentiality (to reverse engineer code and find vulnerabilities), compromise authenticity and integrity (to install backdoored software), or deny availability (to force clients to run outdated vulnerable versions).

To guarantee survivability, the system relies on threshold assumptions, meaning it remains secure as long as the number of compromised actors does not exceed specific limits:

- No more than $t_d - 1$ out of n_d developers are malicious or have had their keys/systems compromised;
- No more than $t_a - 1$ admins and no more than $t_r - 1$ ABE servers are malicious;
- No more than $t_v - 1$ validators and no more than $t_w - 1$ (which equals $\lfloor n_w/3 \rfloor$) witnesses are malicious.

Admins and ABE Servers:

Validators and Witnesses:

Furthermore, we assume the authentication server is honest and that an attacker cannot compromise it. Clients are also assumed to be honest, meaning an attacker cannot easily compromise or impersonate a legitimate vehicle to obtain access to decrypted updates. Orthogonal solutions to prevent the reverse engineering of

the released binaries (e.g., obfuscation or anti-tampering techniques) are outside the scope of this work. Moreover, validators are trusted not to leak the plaintext source code, representing a necessary trade-off between source code confidentiality and transparency. Since the distribution network is treated as entirely untrusted, an attackers may replace, eavesdrop on, or modify the software updates during distribution, though they are assumed to be unable to impede the availability for an unbounded amount of time.

4 Framework design

4.1 Framework components and operations

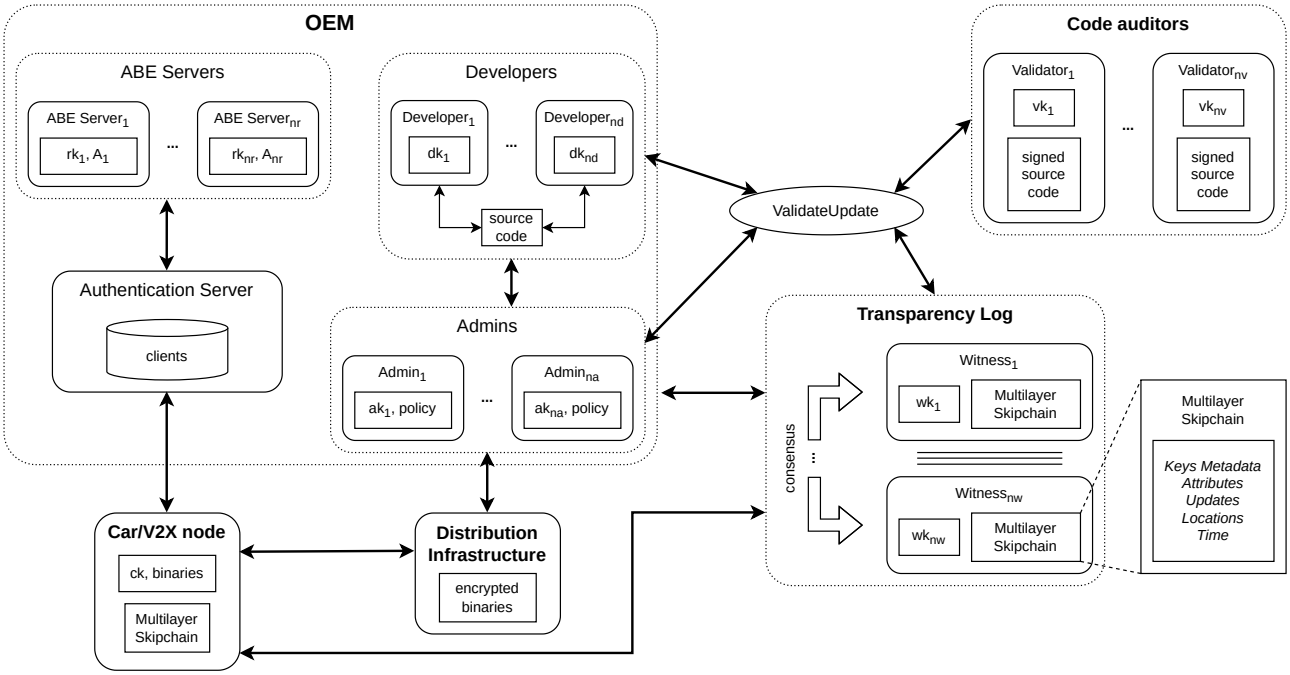

Figure 1: Architecture of the framework for secure software updates in vehicular networks.

Figure 1 shows the components of the proposed framework and their interactions. The secure software distribution framework includes nine operations that manage the update lifecycle, involving several interacting entities distributed across different trust domains. When a new software update is ready for release, the developers and admins of the OEM collaborate to authenticate the source code and metadata of the update. The authenticated update is then validated by independent code auditors, who verify the source-to-binary correspondence and issue an authenticated attestation. Admins then append to the skipchain a new block containing the metadata of the update, including the attestation from the validators, and pubblish the authenticated encrypted binaries to the distribution infrastructure. Clients, who have already obtained their ABE decryption keys from the ABE servers, periodically query the skipchain to check for new updates, and upon detecting one, pull the encrypted binaries from the distribution infrastructure and decrypt them using their ABE keys. Admins can also execute policy updates (e.g., in response to a new update that requires a policy change), key rotation procedures (e.g., in the event of a security incident or key expiration), and change the distribution location (e.g., to change the CDN provider).

4.2 Multi-layer skipchain

To guarantee the freshness and non-equivocation of software update metadata to clients, the witness nodes maintain a specialized data structure called a multi-layer skipchain. This structure extends standard blockchain concepts, allowing administrators to perform survivable and authenticated modifications of the cryptographic material of the system without breaking the chain of trust.

The proposed multi-layer skipchain consists of eight distinct layers, stacked in a specific hierarchical order:

- The Key Metadata layers, the first four layers, are dedicated to manage and store the public keys of

admins, witnesses, validators, and ABE servers. These layers allow a threshold of admins to securely rotate public keys of these actors in the event of a compromise or routine update.

- The Attributes layer maintains the set of ABE attributes that are adopted in the access control policy used to encrypt the latest software update. Admins can add new attributes to this layer when needed if authenticated by a threshold of admins. Because the skipchain is append-only, any attribute or policy changes only affect future updates and have no effect on existing ciphertexts already published.
- The Update layer maintains the critical metadata about the encrypted software binaries. New update metadata can be added to this layer only if it has been authenticated by a threshold of admins, developers, and validators.
- The Location layer contains authenticated location information, such as a URL, which tells the clients how to retrieve the encrypted update binaries from the distribution infrastructure. This layer allows a threshold of admins to change the storage location of the update when necessary.
- The Time layer stores multi-signed timestamped hash pointers to the location layer. These timestamps are periodically computed and appended by the witnesses. Assuming a vehicle has a reliable time source, this layer allows clients to detect freeze attacks: the client compares the most recent timestamp on the skipchain with its current timestamp, and, if the difference exceeds an upper bound (*freshness tolerance*) defined by the admins in the update metadata, the client knows it is being blocked from seeing the true current state of the network.

5 Vehicular software update workflow

To securely deploy software updates to vehicular nodes without relying on a single trusted entity, the framework integrates advanced cryptographic primitives directly into the update lifecycle. Building upon the foundational architectures and cryptographic concepts established in previous Work Package 2 and 3, this section outlines the step-by-step workflow for authenticating, validating, and retrieving a software update.

5.1 Setup and policy definition

Each admin, witness, validator, ABE server and developer generates a key pair and maintains the secret key private. Then, admins act as certification authorities and collectively authenticate the public keys of witnesses, validators and ABE servers through multi-signatures and by assigning validity periods that produce key metadata. The key metatada are stored in the appropriate skipchain layer.

Before an update is released, the OEM must establish the access control rules. Admins define access control policies based on vehicle attributes (e.g., model, subscription tier), compute a new set of attributes and assign each attribute to an ABE server. The original access control policy is translated into a multi-authority policy that relies on some ABE attributes. This ensures that the capability to generate decryption keys is distributed, meaning attackers would need to compromise at least a threshold of ABE servers to violate confidentiality. The multi-authority policy and the corresponding attributes are then appended to the attributes layer of the skipchain so clients can determine which keys are needed (or should be refreshed) to decrypt the next update.

Post-Quantum resilience. To ensure the long-term security of the vehicular network against emerging cryptographic threats, the framework can integrate the post-quantum Ciphertext-Policy Attribute-Based Encryption (CP-ABE) scheme based on Ring Learning with Errors (RLWE) proposed by Gur et al. [4], as explored in WP3. This post-quantum scheme can be seamlessly substituted into the ABE initialization phase to protect the deterministic encryption keys against future quantum computing attacks. This integration is highly viable because the RLWE-based CP-ABE scheme achieves practical performance and manageable overhead even on resource-constrained embedded platforms representative of modern automotive environments. This ensures that the system's enhanced security does not compromise the timeliness or operational efficiency of the vehicular update process.

5.2 Update authentication

When developers are ready to release a new software version, the source code must be approved by code auditors that check its origin and integrity. A threshold of admins authenticates the access control policy, the freshness tolerance of the released software and the set of developers that are authorized to sign the update. A threshold of developers compiles the source code through reproducible build procedures so that all of them obtain the exact same binary data. They collaboratively agree on a shared deterministic encryption key via an authenticated group key agreement and encrypt the update binaries using this key. Each participating developer encrypts the shared deterministic encryption key using the Multi-Authority Ciphertext-Policy Attribute-Based Encryption (MA-CP-ABE) scheme, applying the multi-authority policy defined by the admins. The developers compute cryptographic digests of the source code, the encrypted binaries, and the encryption keys. They use a multi-signature scheme to authenticate this material, producing authenticated update metadata intended to be used by validators in the update validation phase and by clients during the update retrieval phase.

Privacy-preserving authentication via Zero-Knowledge proofs. To protect the OEM from targeted attacks, coercion, or profiling, the update authentication phase can integrate the Identity-Based Authentication paradigm augmented with Zero-Knowledge Proofs (ZKPs), as established in WP2. Instead of exposing explicit identifiers when multi-signing the metadata, developers can act as provers, generating a ZKP that mathematically attests they possess a valid, policy-compliant credential from the software house. This allows the skipchain (the verifier)

to confirm that the required threshold of legitimate developers approved the release without ever revealing the actual underlying credentials or individual identities of the developers. This approach enforces static policies and ensures origin authenticity while actively preserving the privacy of the developers.

5.3 Update validation and publication

To guarantee software transparency and protect against malicious code injection, independent validators audit the update before publication. A designated developer sends the update metadata, the digest of the encrypted binaries, the source code, and the shared deterministic encryption key to the validators over a confidential and authenticated channel. The validators independently compile the source code using reproducible builds, encrypt the resulting binary with the provided shared deterministic encryption key, and compare the resulting digests against the developer-signed metadata. If all checks pass and the source-to-binary correspondence is cryptographically proven, the validators apply their own multi-signature to the authentication material, formally attesting to its validity. Finally, admins send the location information, the authentication material and the validators multi-signature to witnesses to update the multi-layer skipchain. Using a Byzantine-fault-tolerant consensus algorithm, the witnesses agree on the validity of the data and append it to the Update and Location layers of the multi-layer skipchain. Any admin can publish the MA-CP-ABE encrypted binaries to the distribution infrastructure at the location inserted in the multi-layer skipchain.

5.4 Client key refresh

To access an update, a vehicular client must possess the correct cryptographic keys for the corresponding attributes associated with the update. The vehicle authenticates with the Authentication Server, which retrieves the client's original attributes from its database. The Authentication Server then coordinates with a threshold of ABE Servers, which independently generate and return the specific MA-CP-ABE decryption keys corresponding to the vehicle's attributes. This process ensures that even if some ABE servers are compromised, the confidentiality of the updates remains intact as long as the number of compromised servers does not exceed the defined threshold.

5.5 Check for updates and decryption

Vehicles autonomously monitor the network for new software releases and securely install them without blindly trusting a single server. For skipchain traversal, the vehicle periodically requests the latest timestamp from at least a threshold of witnesses to detect and prevent equivocation. It receives the required blocks of the skipchain and uses cryptographic forward and backward links to efficiently validate the update history. By checking the Time layer of the skipchain, the vehicle validates the latest timestamp against its internal timestamp. If the difference is within the freshness tolerance defined by the admins, it proves the vehicle is not the victim of a freeze attack. Upon detecting a new, valid update block, the vehicle downloads the encrypted binaries from the location specified in the skipchain. Using its ABE keys, it decrypts the shared deterministic encryption key, decrypts the software binaries, and safely installs the update.

6 Security analysis

Protection against single points of failure and survivability is guaranteed by the adoption of multi-signatures coupled with validity thresholds across all roles. An adversary cannot forge any authenticated cryptographic material without compromising a minimum threshold of actors (e.g., developers, admins, or witnesses). To manipulate the public update timeline managed by the update subgroup of signers, an attacker needs to compromise at least a threshold of witness servers. Similarly, an attacker must compromise a threshold of build verifiers to break the source-to-binary release correspondence. The key rotation procedure ensures recoverability. In fact, if a threshold of admins, witnesses, verifiers, ABE servers or developers keys are compromised, then admins can recover the system to a safe state by rotating the compromised keys as soon as the incident is detected.

Confidentiality of proprietary code is protected against an adversary who intercepts the binaries, either by compromising the distribution infrastructure or by eavesdropping on them over the network. Because the deterministic encryption key is wrapped using MA-CP-ABE, the adversary must violate at least a threshold of ABE servers to be able to obtain the decryption key. This ensures that the compromise of a single ABE server does not result in a data breach. The confidentiality of the software update's source code relies on the assumption that validators and developers are trustworthy. Attacks by one developer could be even minimized by adopting software management techniques that segment source code and prevent one developer to access the whole code and/or by detailed logging and forensics mechanisms, but the integration of similar solutions is out of the scope of this work.

Authenticity of software updates is protected by admins and developers who digitally multi-sign software updates data and metadata. **Transparency** is guaranteed by validators who check the update validation procedure. By requiring reproducible builds, the update subgroup of signers will refuse to sign a release proposal whose binaries differ from those reproduced by the independent build verifiers. This allows validators to detect attacks where an adversary induces developers into signing backdoored software update binaries that do not correspond to the original source code. **Integrity** is guaranteed by the multi-layer skipchain, which provides an immutable ledger of update metadata and location information. The skipchain's backward links are cryptographic hashes, ensuring the immutability of the past with respect to any valid release. An attacker attempting to fork the log will fail because of the consensus mechanism.

Timeliness of updates is protected by the Time layer of the skipchain, which allows clients to detect freeze and replay attacks. The adoption of a consensus protocol guarantees that clients have a consistent view of the skipchain state. By querying at least t_w witnesses, the client ensures at least one honest witness is queried, which will expose any equivocation attacks by malicious nodes returning inconsistent responses. Also, the Time layer of the skipchain allows clients to detect freeze attacks. Assuming the vehicle has a correct internal timestamp, it can detect freeze and replay attacks by verifying the collectively signed timestamps. If an attacker controls the vehicle's network connection and presents a stale view of the skipchain, the missing fresh time updates will alert the vehicle to the attack.

7 Conclusions

The increasing complexity and connectivity of modern vehicular networks demand a paradigm shift in how software updates are managed and distributed. Traditional centralized update systems are vulnerable to single points of failure and lack the necessary transparency, confidentiality, and survivability required for critical automotive infrastructure.

The framework proposed in this document addresses these critical gaps by uniting two powerful cryptographic paradigms: the transparent immutable logging of multi-layer skipchains and the robust access control of Multi-Authority Attribute-Based Encryption. This combination yields a highly resilient architecture specifically tailored for the secure distribution of proprietary software updates. By distributing authority across multiple roles the system guarantees survivability, and remains secure and operational even if a threshold of the system actors is maliciously compromised. The mandatory use of reproducible builds by independent validators ensures that the compiled binaries distributed to vehicles cryptographically match the approved source code, thwarting stealthy code injection. The skipchain's decentralized time layer and consensus mechanisms actively protect resource-constrained vehicular clients against targeted freeze, rollback, and equivocation attacks. And finally, in the event of a detected compromise, the framework allows administrators to rapidly rotate keys and restore the system to a safe state without service disruption or requiring a complete system reset.

References

[1] N. Asokan, Thomas Nyman, Norrathep Rattanavipanon, Ahmad-Reza Sadeghi, and Gene Tsudik. Assured: Architecture for secure software update of realistic embedded devices. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2018.

[2] Amrita Ghosal, Subir Halder, and Mauro Conti. Stride: Scalable and secure over-the-air software update scheme for autonomous vehicles. In *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*, 2020.

[3] Amrita Ghosal, Subir Halder, and Mauro Conti. Secure over-the-air software update for connected vehicles. *Computer Networks*, 2022.

[4] Kamil D. Gür, Yuriy Polyakov, Kurt Rohloff, Gerard W. Ryan, Hadi Sajjadpour, and Erkay Savaş. Practical applications of improved gaussian sampling for trapdoor lattices. *IEEE Transactions on Computers*, 2019.

[5] Trishank Karthik Kuppusamy, Lois Anne DeLong, and Justin Cappos. Uptane: Security and customizability of software updates for vehicles. *IEEE Vehicular Technology Magazine*, 2018.

[6] Michele La Manna, Luigi Treccozzi, Pericle Perazzo, Sergio Saponara, and Gianluca Dini. Performance evaluation of attribute-based encryption in automotive embedded platform for secure software over-the-air update. *Sensors*, 2021.

[7] Federico Magnanini, Luca Ferretti, and Michele Colajanni. Scalable, Confidential and Survivable Software Updates . *IEEE Transactions on Parallel & Distributed Systems*, 2022.

[8] Kirill Nikitin, Eleftherios Kokoris-Kogias, Philipp Jovanovic, Nicolas Gailly, Linus Gasser, Ismail Khoffi, Justin Cappos, and Bryan Ford. CHAINIAC: Proactive Software-Update transparency via collectively signed skipchains and verified builds. In *26th USENIX Security Symposium*, 2017.

[9] Ewa Syta, Iulia Tamas, Dylan Visher, David Isaac Wolinsky, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ismail Khoffi, and Bryan Ford. Keeping authorities "honest or bust" with decentralized witness cosigning. In *2016 IEEE Symposium on Security and Privacy (SP)*, 2016.