Current **A**pplied **S**cience and **T**echnology e-ISSN: 2586-9396

Curr. Appl. Sci. Technol. 2025, Vol. 25 (No. 6), e0257532

Research article

Security Analysis of Biometric Verification Protocol Using Scyther Model Checker: A Case Study

Pradeep Rajanna* and Nagarajaiah Renukamba Sunitha

Department of Computer Science and Engineering, Siddaganga Institute of Technology,Tumkur - Karnataka, 572103, India, Affiliated to Visvesvaraya Technological University, Belagavi - Karnataka, 590018, India

Received: 24 January 2023, Revised: 3 February 2025, Accepted: 22 March 2025, Published: 8 May 2025

Abstract

Aadhaar is a biometric-based national identification system used in India that relies on fingerprints, iris scans, and face images as primary means of verification. Ensuring the reliability and security of biometric protocols is critical, as they must be robust and resistant to attacks. To achieve and prove the reliability of a biometric security protocol, the traditional testing technique is not a desirable solution because of its limitations. The formal verification technique is a better solution to prove the reliability of a biometric security protocol as it provides mathematical proofs to validate security protocols. The biometric capture devices used in the Aadhaar system must comply with either Level-0 (L0) or Level-1 (L1) security standards as defined by the Unique Identification Authority of India (UIDAI). The L0 protocol is widely adopted due to its cost-effectiveness, though it is less secure compared to the L1 protocol. This paper focuses on the formal analysis and verification of the L0 iris-based biometric verification security protocol. Using the Scyther model checker, security vulnerabilities in the L0 protocol are identified. A security solution is then proposed by addressing these vulnerabilities and formally proving the correctness of the improved security model against the Dolev-Yao adversary model.

Keywords: biometric; Aadhar; security; L1 protocol; formal verification; Scyther

1. Introduction

The Biometric authentication protocols (Sireesha & Reddy, 2016) are a specific type of security protocol that uses unique physical or behavioral traits to verify user identity. Key biometrics include face, iris, voice, vein pattern, and even gait. Biometric verification protocols (Jain et al., 2022) must be highly secure when communicating biometric data over an insecure network, as biometric data are highly confidential and harder to replicate or forge than conventional authentication methods such as passwords or security tokens (Sharma et al., 2023). The primary benefit of biometric security protocols is that biometrics cannot be shared, unlike passwords. Additionally, they spare users from remembering or managing other credentials. A variety of biometrics-based authentication protocols are applied globally, each with advantages and disadvantages. Metrics such as accuracy,

*Corresponding author: E-mail: pradeepr@sit.ac.in
https://doi.org/10.55003/cast.2025.257532

measurement speed, and cost determine their suitability for specific tasks. Biometric security protocols are not perfect or ideally secure, despite their benefits. Privacy concerns and the potential for misuse or exploitation of biometric data (Adeyanju et al., 2021) are significant challenges. Some biometric features are more vulnerable to impersonation or hacking than others. Most biometric authentication systems, such as Aadhaar (Singh & Jackson, 2021) and MOSIP (Sheik et al., 2022), which handle large databases, use a client-server model where users' biometrics are stored in centralized data centers. Aadhaar primarily uses two biometric security protocols for biometric authentication, Level-0 L0 and Level-1 L1, which represent different levels of biometric template protection when communicating over insecure channels. L0 protocols have less significant security enhancements for biometric data, making the biometric templates vulnerable to attacks such as spoofing, replay, or tampering. L1 protocols, in contrast, use primitive security enhancement techniques such as biometric data encryption, hashing, and template transformation. The L1 attempts to obfuscate compromised biometric templates so attackers cannot reverse-engineer or misuse them. The L1 protocol uses hardware encryption technique (Bao et al., 2024) in which the captured biometric template is encrypted by the key, which is pre-stored in the hardware chip. The UIDAI has pre-certified this key.

Biometric security protocols (Adeyanju et al., 2021) are critical protocols widely used in access control systems and KYC applications by verifying user identity based on biometric traits. Biometric protocols must be highly reliable and free from attacks. To achieve reliability, testing technique is not a desirable solution because of its limitations. One such limitation is the tester cannot predict attacker's capabilities. Formal verification technique overcomes all the limitations of testing technique, and it helps to prove the reliability of biometric security protocols by providing mathematical proofs. Formal verification technique not only shows the presence of attacks but also proves the absence of security attacks. Despite significant progress, formal verification of security protocols faces multiple ongoing challenges. One major issue is the state explosion problem (Jiang et al., 2021), which arises due to the vast number of possible protocol states, especially in complex or real-world protocols. Researchers have explored abstraction techniques and compositional verification to address this challenge, but scalability remains a concern. Moreover, formal verification often assumes idealized models of cryptographic primitives, which may not accurately reflect real-world implementations.

The advancement of formal verification (Lewis et al., 2023) technology in the biometric security protocol domain, specifically the model-checking technique, addressed the strong need for proving the reliability of secure communication protocols. However, as the protocols became more complex, informal security analysis methods, such as legacy testing techniques and manual reviews, often failed to detect design flaws and security vulnerabilities. The application of formal methods to the analysis of security protocols marked a significant shift in the early 1990s. Burrows, Abadi, and Needham's seminal work (BAN logic) (Thakur et al., 2024), which introduced a formal reasoning system specifically for authentication security protocols, paved the way. BAN logic, although limited in scope, highlighted the potential of the use of formal verification methods to rigorously analyze and verify security properties of protocols. Over time, formal verification approaches evolved by incorporating techniques from model checking, theorem proving, and symbolic analysis. Model-checking methods (Pradeep & Sunitha, 2022) gained popularity in verifying finite-state protocols by exhaustively exploring the protocol state space, which encompasses all possible behaviors of the protocol. The development of automated and advanced tools further increased the strength of formal verification techniques (Krichen, 2023). One such tool is FDR (Failures-Divergence Refinement), which is used for verifying CSPs

(Cryptographic Security Protocols). FDR exponentially increased the practical applicability of formal methods in security protocol verification. Formal verification of security protocols primarily relies on the development of abstract security protocol models that represent protocol behavior and interaction with adversaries. These models vary depending on the type of protocol and the desired security properties, such as confidentiality, integrity, or authentication that must be satisfied by the model under verification.

Process algebras such as CSP and Pi-calculus (Baillot & Ghyselen, 2022) are widely used to model the communication and concurrency aspects of security protocols. These mathematical frameworks allow for formal reasoning about the interactions between protocol entities and adversaries. For instance, in CSP, model checkers such as FDR can express security properties as refinements and verify these properties against protocol specifications. Symbolic analysis abstracts cryptographic operations by treating them as black boxes, allowing verification without delving into cryptographic primitives' details. The Dolev-Yao (Rakotonirina et al., 2024) model is an example of symbolic analysis, where messages are represented symbolically, and adversaries are assumed to have complete control over the communication channel, except for the ability to break cryptography. Tools such as ProVerif (Blanchet et al., 2022) have built on this model, enabling automatic verification of security properties through symbolic execution. Security protocols have adopted tools like SPIN (Garanina et al., 2023) and Murphi (Cai et al., 2024) for verification. The model-checking methods systematically explore the protocol's state space to ensure that security properties are held in all possible states. Although model checking is exhaustive, it faces challenges with scalability due to the state explosion problem. Techniques like abstraction and symmetrical reduction are often employed to mitigate this issue.

Theorem Proving (Yang et al., 2023): Unlike model checking, theorem proving requires the manual construction of formal proofs demonstrating that a protocol satisfies certain properties. Isabelle and Coq are environments used to develop and mechanically check such proofs. Theorem proving provides more generality and works for infinite-state systems, but it is much harder and requires expertise in the area. The industry standard formal verification tools for formal analysis and verification of security protocols are: 1) ProVerif - a tool that performs symbolic verification of security properties such as secrecy and authenticity in cryptographic protocols. ProVerif uses the Dolev-Yao model and can reason about protocols in the presence of active adversaries. 2) Tamarin (Celi et al., 2022) - a verification tool that combines symbolic reasoning with equational theories for handling cryptographic operations. It verifies a broad class of security properties, including indistinguishability and trace properties. 3) AVISPA (Ram et al., 2024) - the Automated Validation of Internet Security Protocols and Applications is a framework for analysing security protocols and provides several back-end solvers for protocol verification. 4) Scyther (Le et al., 2024) – a model checker specially designed for formal analysis and verification of security protocols. Scyther analyses protocols for vulnerabilities such as authentication failures or data leakage by modelling interactions and attack scenarios. It provides automated analysis, visual representations, proofs or counterexamples and handles state-explosion to ensure protocol security, which makes it the best tool for formal analysis of biometric security protocols.

2. Materials and Methods

In this work, we provide the first independent examination of the L0 Aadhar iris-based verification models. This study demonstrates a successful compromise of biometric data

confidentiality. We introduce an innovative protocol for Aadhar, namely the Secure L0 Iris Biometric Verification Protocol (SLIBAP), and provide the findings of the assessment together with the details of a prototype implementation. Using GNY logic and the Scyther tool, we prove the security of the proposed protocol and ascertain that SLIBAP offers the necessary defense against diverse passive and active cryptographic attacks. The Scyther model checker tool (v1.1.3), a formal verification instrument for security protocols that leverages the Dolev-Yao attacker model to simulate an active opponent, was employed. Python 2.7, Graphviz version 6.0.2, and wxPython 2.8 were used for the creation of attack graphs.

Figure 1 shows the secure, biometric-based authentication using a combination of biometric devices, Aadhar infrastructure, and secure key management techniques.

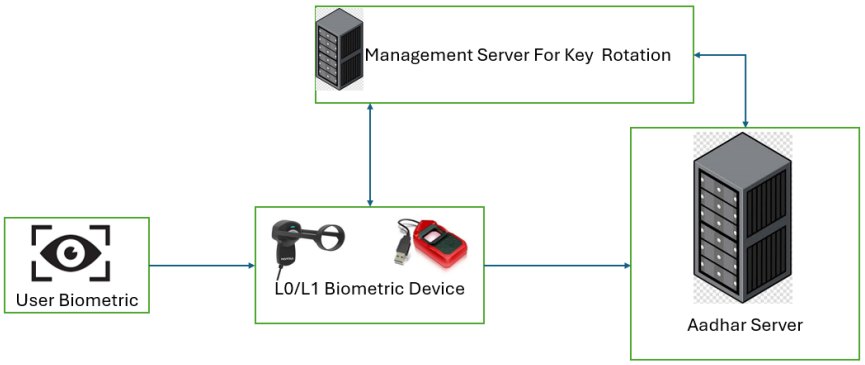

Figure 1. Aadhar L0 IRIS verification protocol stakeholders

L0/L1 biometric devices: These devices collect the user's biometric data (e.g., fingerprint, iris scan) and send it for authentication. Biometric devices were designed in 2 categories: 1) enrolment devices and 2) authentication devices. L0 devices represent the level-0 and refer to the most basic level of biometric devices used in the Aadhar authentication ecosystem. These devices are typically used for capturing biometric data, such as fingerprints or iris scans, and transmitting a biometric template for authentication purposes. These devices are used in applications where biometric capture is required, but real-time remote authentication is sufficient, such as public distribution systems, financial services, and identity verification processes. L0 devices capture biometric data such as fingerprints with minimal processing done on the device itself, and no advanced security mechanism such as hardware encryption were included. The device is primarily responsible for data acquisition. The L0 device does not perform local biometric matching or authentication. Instead, it sends the captured biometric data to an external server, typically the UIDAI Aadhar Authentication Server, where the verification is done against the central Aadhar database. The L0 have lower security and processing requirements compared to L1 devices. The key stakeholders in Aadhar ecosystem are as follows:

User biometrics: The user biometric characteristics, such as fingerprints or iris scans, that are being captured using L0 or L1 devices for authentication.

Aadhar server: This is a centralized server that verifies the biometric data received against its biometric template stored in the database for identity verification. It plays a key role in authenticating users based on their Aadhar demographics.

Management server for key rotation: This server handles cryptographic keys used in securing communication between the biometric devices and the Aadhar server. It mainly handles regular key rotation to ensure enhanced security for the biometric authentication transactions.

2.1 Security properties tested

Security protocols are mainly designed to ensure various security properties that protect communication and biometric data. These properties can be formally verified to assess the security and integrity of the protocol under verification. Common security properties that can be verified with a security protocol model include:

Confidentiality: Confidentiality is the primary property in any security protocol to ensure that sensitive information remains accessible only to intended parties. It protects data from unauthorized access or disclosure, maintaining privacy and trust. Encryption techniques, secure communication channels, and strict access controls achieve this, safeguarding confidential data from eavesdropping and cyber threats from attacker.

$$\forall\ adversary\ A, Pr[A(C) = M]\ \leq \epsilon \tag{1}$$

Where: *A* represents the active adversary, C represents the encrypted ciphertext, *M* represents the plaintext message, and *ϵ* is a negligible probability.

Integrity: The integrity of a security protocol ensures that data is accurate, consistent, and unaltered by the communicating roles during transmission or storage. It protects against unauthorized modifications, ensuring that the information received is exactly as intended. We employ techniques such as hashing, checksums, and digital signatures to verify data integrity, prevent tampering, and ensure reliability.

$$\forall\ adversary\ A, Pr[A(C')\ \rightarrow M\ and\ M' \neq M]\ \leq \epsilon \tag{2}$$

Where: *A* represents the adversary, *C'* represents the modified ciphertext, *M* represents the original plaintext message, *M'* represents the adversary's altered plaintext, and ϵ is a negligible probability.

Authentication: In security protocols, the authentication property verifies the identity of the users or systems before granting access to the network resources. It ensures that only legitimate entities can interact with the system after verifying their identities. It is primarily achieved using passwords, biometrics, multi-factor authentication, and digital certificates. Let us consider a message *M* and the recipient *R*:

$$\forall adversary\ A,\ \Pr\left[A\ successfully\ convinces\ R\ that\ a\ message\ M\ originated\ from\ a\ legitimate\ sender\ S\right] \leq \epsilon \tag{3}$$

Where: *A* represents the adversary, *M* represents the original plaintext message*, R* represents the recipient, and ε is a negligible probability.

Non-repudiation: The non-repudiation property of a security protocol ensures that a party cannot deny the authenticity of their actions or communications. It provides proof of origin and integrity of the messages sent, often through digital signatures and audit trails. This guarantees that transactions and messages are verifiable and legally binding, preventing parties from falsely denying involvement.

$$\forall sender\ S\ and\ recipient\ R, Pr\ [S\ denies\ having\ sent\ message\ M\ to\ R, \\ but\ R\ can\ prove\ M\ was\ sent\ by\ S] \geq 1 - \varepsilon \quad (4)$$

Where: *S* represents the sender, *R* represents the recipient, *M* represents the message, and *ε* is a negligible probability.

2.2 Dolev-Yao (DY) attacker model

The Dolev-Yao adversary is the strongest attacker model used in the formal verification of security protocols. This adversary is active, with the ability to intercept, modify, inject, and delete messages sent over an insecure network. The model assumes that the adversary has complete access to the network and can manipulate any message passing through it, as shown in Figure 2. This includes the ability to eavesdrop on every communication, alter the content of messages, and fabricate new messages entirely. Despite these capabilities, the Dolev-Yao adversary is constrained by the cryptographic primitives in use. Specifically, the adversary cannot break encryption schemes, generate valid digital signatures without the correct private key, or solve complex mathematical problems that underlie cryptographic algorithms, such as factoring large numbers or computing discrete logarithms. The Dolev-Yao adversary model assumes that the adversary has several significant capabilities:

Eavesdropping: The attacker has access to any data sent across the network and can listen in on all conversations between participants.

Message modification: The adversary can modify communications in transit, thereby altering the content, origin, or destination of the communication. This capability is essential for the simulation of real-world scenarios in which adversaries may attempt to manipulate data.

Message injection: The adversary can introduce new communications into the network by assuming the identity of a legitimate party. This enables the adversary to evaluate the protocol's capacity to differentiate between authentic and forged messages.

Message deletion: By deleting messages, the adversary can obstruct them from reaching their intended recipients. This capability simulates denial-of-service attacks or other forms of communication disruption.

Message replay: The adversary can store messages and replay them at a later time, potentially causing confusion or unauthorised actions if the protocol does not account for such attacks.

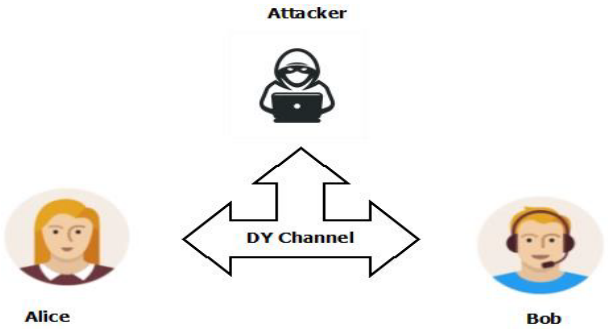

Figure 2. Roles communicating over the DY channel

The Scyther model checker uses the DY attacker model to prove the security properties of the protocol under verification. Figure 3 shows the L0 protocol abstract security model. Table 1 gives the description of parameters used to build the L0 protocol abstract model.

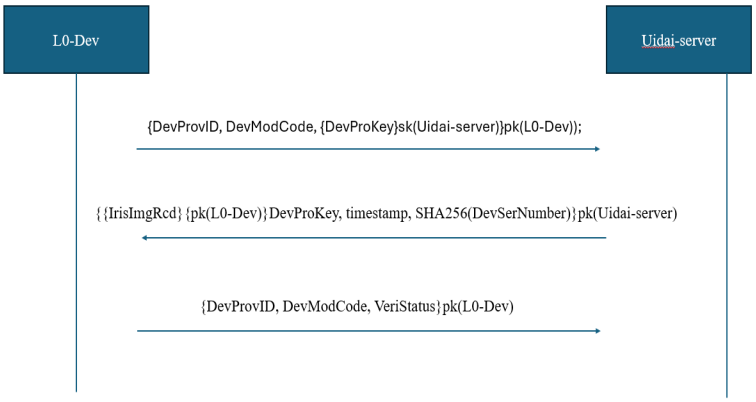

Figure 3. Existing UIDAI L0 IRIS verification protocol steps

Table 1. Notations used in the L0 protocols' description

Acronym	Meaning
L0-Dev	*Device in Level 0 of biometric security protocol*
Uidai-server	*Unique Identification Authority of India (UIDAI) Server*
DevSerNumber	*Device Serial Number*
IrisImgRcd	*Iris Image Record*
DevProvID	*Device Provider ID*
DevModCode	*Device Model Code*
DevProKey	*Device Provision Key (Session Key)*
SHA256	*Secure Hash Algorithm 256-bit*
timestamp	*Time reference used in the protocol*
nonce	*Number used once in a cryptographic context*
VeriStatus	*Verification Status*
sk(Uidai-server)	*UIDAI Server's Private Key*
pk(L0-Dev)	*L0 Device's Public Key*
pk(Uidai-server)	*UIDAI Server's Public Key*
Nisynch	*Non-injective synchronization*
Niagree	*Non-injective agreement*

2.3 Existing L0 IRIS verification protocol

Model 1- Existing Aadhar L0 IRIS verification protocol model.

```
usertype SessionKey;
usertype String;
hashfunction SHA256;
usertype Imgrcd;
usertype Binary;
usertype timestamp;

protocol Aadhar(L0-Dev, Uidai-server) {
   role L0-Dev {
      fresh DevSerNumber: String;
      fresh IrisImgRcd: Imgrcd;
      var DevProvID: String;
      var DevModCode: String;
      var DevProKey: SessionKey;
      fresh Timestamp: timestamp;
      var nonce1: Nonce;
      fresh nonce2: Nonce;
      var VeriStatus: Binary;

      recv_1(Uidai-server, L0-Dev, {
         DevProvID, DevModCode, {DevProKey}sk(Uidai-server)
      }pk(L0-Dev));

      send_2(L0-Dev, Uidai-server, {{IrisImgRcd}{pk(L0-Dev)}DevProKey, timestamp,
SHA256(DevSerNumber)}pk(Uidai-server));

      recv_3(Uidai-server, L0-Dev, {
         DevProvID, DevModCode, VeriStatus
      }pk(L0-Dev));

      // Security claims
      claim(L0-Dev, Secret, timestamp);
      claim(L0-Dev, Secret, IrisImgRcd);
      claim(L0-Dev, Secret, DevSerNumber);
      claim(L0-Dev, Secret, VeriStatus);
      claim(L0-Dev, Secret, DevProKey);
      claim(L0-Dev, Secret, DevModCode);
      claim(L0-Dev, Secret, DevProvID);
      claim(L0-Dev, Secret, sk(Uidai-server));
      claim(L0-Dev, Alive);
      claim(L0-Dev, Weakagree);
      claim(L0-Dev, Nisynch);
      claim(L0-Dev, Niagree);
   }

   role Uidai-server {
```

```
      fresh DevProvID: String;
      fresh DevModCode: String;
      fresh DevProKey: SessionKey;
      var DevSerNumber: String;
      var IrisImgRcd: Imgrcd;
      var timestamp: timestamp;
      fresh nonce1: Nonce;

      send_1(Uidai-server, L0-Dev, {
         DevProvID, DevModCode, {DevProKey}sk(Uidai-server)
      }pk(L0-Dev));

      claim(Uidai-server, Secret, DevProKey);

      var nonce2: Nonce;
      recv_2(L0-Dev, Uidai-server, {{IrisImgRcd}{pk(L0-Dev)}DevProKey, timestamp,
SHA256(DevSerNumber) }pk(Uidai-server));

      fresh VeriStatus: Binary;
      send_3(Uidai-server, L0-Dev, {
         DevProvID, DevModCode, VeriStatus
      }pk(L0-Dev));

      // Security claims
      claim(Uidai-server, Secret, DevProKey);
      claim(Uidai-server, Secret, DevSerNumber);
      claim(Uidai-server, Secret, VeriStatus);
      claim(Uidai-server, Secret, DevProvID);
      claim(Uidai-server, Secret, sk(Uidai-server));
      claim(Uidai-server, Secret, IrisImgRcd);
      claim(Uidai-server, Secret, timestamp);
      claim(Uidai-server,Nisynch);
      claim(Uidai-server, Niagree);
      claim(Uidai-server, Alive);
   }
}
```

When the existing L0 protocol was formally verified using the Scyther model checker, the security attacks listed in Table 2 were detected, indicating that the existing protocol was not fully secure. The attacks on L0 protocol and its solutions are as follows:

1. Replay attacks:

- Attack: An attacker could capture valid messages exchanged between the L0-Device and UIDAI server and replay them at a later time.
- Impact: The UIDAI server or L0-Device could accept old, valid messages, believing they are fresh. This could lead to false authentication of stale or compromised biometric data.
- Prevention: Introduce nonces or timestamps in all messages and ensure that they are validated for freshness by both parties.

Table 2. Attacks detected in the L0 protocol

Attack Type	Attack Present
Replay Attack	✓
Man-in-the-Middle (MitM) Attacks	✓
Key Compromise	✓
Insider Attacks	✓
Biometric Data Theft	✓
Collision Attack	✓
Denial of Service (DoS) Attacks	✓

2. Man-in-the-Middle (MitM) attacks:

- Attack: An attacker could intercept communication between the L0-Device and the UIDAI server and alter the contents of messages, especially if they can break the encryption.
- Impact: The attacker could modify data such as the IrisImgRcd, DevSerNumber, or VeriStatus and send it to the other party, leading to false identification or authentication.
- Prevention: The current protocol encrypts sensitive data with public and private keys, but adding message authentication codes (HMAC) or signatures on all critical parts of the message could provide integrity checks, making it harder for attackers to modify messages undetected.

3. Key compromise:

- Attack: If the shared session key (DevProKey) or the UIDAI server's private key (sk(Uidai-server)) is compromised, an attacker can decrypt and re-encrypt messages, or impersonate either of the communicating roles.
- Impact: The attacker could completely take over communication, leading to unauthorised access, replay of biometric template data, or impersonation.
- Prevention: Use session key rotation and possibly a more frequent exchange of fresh keys. Ensuring that keys are not reused for extended periods can help limit the impact of a key compromise attack.

4. Insider attacks:

- Attack: An insider at the UIDAI server could tamper with VeriStatus or other sensitive data and send false verification results.
- Impact: The L0-Device would receive fraudulent verification results, potentially leading to unauthorized access for a verified person.
- Prevention: Use accountability mechanisms like logs or non-repudiation techniques to ensure that every action within the server can be traced to an authenticated user.

5. Biometric data theft:

- Attack: If the IrisImgRcd (biometric data) is intercepted or extracted from the L0-Device or UIDAI server by an attacker, the attacker could use this stolen biometric template information for future authentication for that verified person and perform biometric based transactions.
- Impact: Stolen biometric data could be reused, compromising the authentication process and privacy of the user.
- Prevention: Ensure that biometric data is properly encrypted during transmission ({IrisImgRcd}{pk(L0-Dev)}) and apply encryption standards that are resilient

against known attacks. Using homomorphic encryption techniques for biometric data could also mitigate the impact of data interception by allowing computations on encrypted data without decryption.

6. Weakness in hashing (collision attack):

- Attack: If the hash function used to hash the DevSerNumber is weak or compromised, an attacker could find two different inputs that hash to the same value (collision attack).
- Impact: The attacker could send a different serial number that produces the same hash, leading the server to believe it's valid, which would weaken the integrity of the protocol.
- Prevention: Ensure that the hash function is secure and resilient to collision attacks. SHA-256 is considered secure at present, but this could change in the future. Regularly update to stronger cryptographic algorithms when necessary.

7. Nonce reuse:

- Attack: If the nonces (nonce1, nonce2) are not generated uniquely for each session or reused across different sessions, an attacker could replay or predict messages.
- Impact: Reusing nonces would make it easier for attackers to inject or replay previous valid messages.
- Prevention: Ensure that nonces are fresh and unique for each session and use larger nonce values (128-bit or more) to prevent collisions or predictability.

8. Lack of mutual authentication:

- Attack: The protocol may only authenticate one party (e.g., the L0-Device), leaving the other party (e.g., UIDAI server) vulnerable to impersonation attacks.
- Impact: The L0-Device might send sensitive data (biometric or otherwise) to a fake UIDAI server.
- Prevention: Implement mutual authentication where both the L0-Device and UIDAI server authenticate each other before exchanging sensitive data.

9. Denial of service (DoS) attacks:

- Attack: An attacker could send repeated invalid authentication requests to the UIDAI server or L0-Device, overwhelming the system and preventing legitimate users from authenticating.
- Impact: The server or device could be made unavailable, leading to a denial of service for legitimate users.
- Prevention: Implement rate-limiting, CAPTCHA, or similar mechanisms to mitigate the impact of repeated malicious requests. Adding resource checks to reject obviously invalid data before consuming too many resources is also crucial.

2.3 Securing L0 verification protocol

The potential vulnerabilities identified; the existing model can be fixed using the cryptographic techniques below.

- Nonce and timestamp validation: Ensure that all nonces and timestamps are validated for freshness and are unique.
- Mutual authentication: Ensure that both parties authenticate each other, not just one side.
- Use of HMAC/Signatures: Include signatures or HMAC on critical parts of messages to ensure integrity and authenticity.

- Regular key rotation: Rotate session keys after a set number of sessions or time to limit the impact of key compromises.

The model 2, as shown in Figure 4, is the fixed version of the L0 protocol.

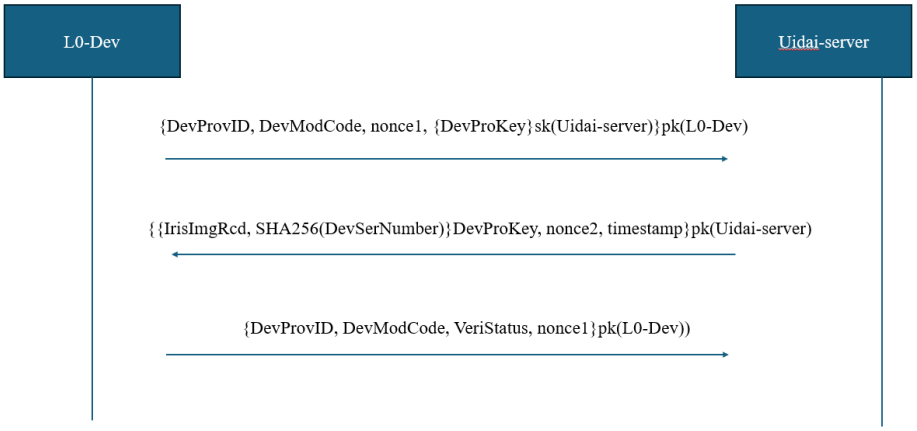

Figure 4. Proposed secure L0 IRIS verification protocol steps

Model 2 Proposed secure L0 verification protocol model

```
usertype SessionKey;
usertype String;
hashfunction SHA256;
usertype Imgrcd;
usertype Binary;
usertype timestamp;

protocol Aadhar(L0-Dev, Uidai-server) {
   role L0-Dev {
      fresh DevSerNumber: String;
      fresh IrisImgRcd: Imgrcd;
      var DevProvID: String;
      var DevModCode: String;
      var DevProKey: SessionKey;
      fresh Timestamp: timestamp;
      var nonce1: Nonce;
      fresh nonce2: Nonce;
      var VeriStatus: Binary;

      # Receive initial parameters from server, including nonce1 for freshness
      recv_1(Uidai-server, L0-Dev, {
         DevProvID, DevModCode, nonce1, {DevProKey}sk(Uidai-server)
      }pk(L0-Dev));
      # Send biometric data, DevSerNumber hash, and freshness (timestamp and
nonce2)
```

```
    send_2(L0-Dev, Uidai-server, {
       {IrisImgRcd, SHA256(DevSerNumber)}DevProKey, nonce2, timestamp
    }pk(Uidai-server));

    # Receive verification status with nonce validation
    recv_3(Uidai-server, L0-Dev, {
       DevProvID, DevModCode, VeriStatus, nonce1
    }pk(L0-Dev));

    # Security claims
    claim(L0-Dev, Secret, timestamp);
    claim(L0-Dev, Secret, IrisImgRcd);
    claim(L0-Dev, Secret, DevSerNumber);
    claim(L0-Dev, Secret, VeriStatus);
    claim(L0-Dev, Secret, DevProKey);
    claim(L0-Dev, Secret, DevModCode);
    claim(L0-Dev, Secret, DevProvID);
    claim(L0-Dev, Secret, sk(Uidai-server));
    claim(L0-Dev, Alive);
    claim(L0-Dev, Weakagree);
    claim(L0-Dev, Nisynch);
    claim(L0-Dev, Niagree);
}

role Uidai-server {
    fresh DevProvID: String;
    fresh DevModCode: String;
    fresh DevProKey: SessionKey;
    var DevSerNumber: String;
    var IrisImgRcd: Imgrcd;
    var timestamp: timestamp;
    fresh nonce1: Nonce;
    fresh nonce2: Nonce;

    # Send initial message with nonce1 for freshness
    send_1(Uidai-server, L0-Dev, {
       DevProvID, DevModCode, nonce1, {DevProKey}sk(Uidai-server)
    }pk(L0-Dev));

    claim(Uidai-server, Secret, DevProKey);

    # Receive biometric data and nonce2, validate nonce for freshness
    recv_2(L0-Dev, Uidai-server, {
       {IrisImgRcd, SHA256(DevSerNumber)}DevProKey, nonce2, timestamp
    }pk(Uidai-server));

    # Generate verification status
    fresh VeriStatus: Binary;
    # Respond with verification status and nonce1 to ensure freshness
    send_3(Uidai-server, L0-Dev, {
```

```
        DevProvID, DevModCode, VeriStatus, nonce1
    }pk(L0-Dev));

    # Security claims
    claim(Uidai-server, Secret, DevProKey);
    claim(Uidai-server, Secret, DevSerNumber);
    claim(Uidai-server, Secret, VeriStatus);
    claim(Uidai-server, Secret, DevProvID);
    claim(Uidai-server, Secret, sk(Uidai-server));
    claim(Uidai-server, Secret, IrisImgRcd);
    claim(Uidai-server, Secret, timestamp);
    claim(Uidai-server, Nisynch);
    claim(Uidai-server, Niagree);
    claim(Uidai-server, Alive);
  }
}
```

3. Results and Discussion

Based on the verification results presented in Figures 5 and 6, the findings demonstrate that the model-checking technique effectively identifies the attacks inherent in the Aadhar L0 security protocol. According to the existing L0 protocol specification, the L0 protocol is less secure than the L1 protocol. This work proposes a possible fix to the existing attacks and introduces a new model that is attack-free. Figures 5 and 6 illustrate how the proposed Model-2 resolves the attacks identified in the existing Model-1. The data in Table 3 presents the model-checking verification time consumed by both the existing and proposed models. The existing model requires less time compared to the proposed model, as this is expected because as the number of variables and cryptographic operations increases, the verification time increases exponentially because of increased numbers of states and transitions in the protocol state space.

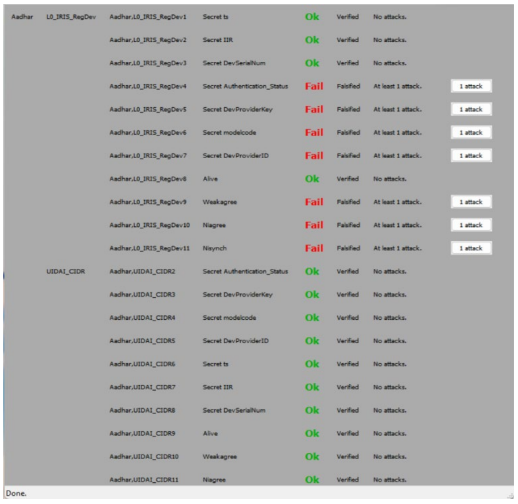

Figure 5. Existing system L0 IRIS verification protocol result

Figure 6. Secure L0 IRIS verification protocol result

Table 3. Verification time consumed by L0 existing and proposed system

Number of Rounds	Verification Times in Seconds	
	L0 Existing	L0 Proposed
1	1	5
6	9	15
10	22	32
15	46	53
20	59	69
25	74	89

4. Conclusions

This study explores the formal analysis and verification of the L0 Iris biometric security protocol for the Aadhar biometric infrastructure. Model checking the current Aadhar L0 biometric verification security protocol revealed the possible security vulnerabilities against the Dolev-Yao adversary, which is the most powerful active adversary model. The verification results showed that when the iris biometric data was sent over the DY channel, there existed replay attacks, MitM attacks, key compromise, insider attacks, biometric data theft, collision attacks, and DoS attacks. The attacks were fixed by proposing a new fixed model for the L0 iris biometric protocol, which incorporated cryptographic techniques such

as the addition of nonce values and role identitty communication. Finally, the proposed L0 iris model was proved secure using the Scyther model checker.

This study showed the effectiveness of the formal verification technique in formal analysis and verification of the L0 biometric security protocol. The experimental results show how the model checking technique can be used to mathematically prove the reliability of the L0 biometric security protocols. This work can be further extended to formal analysis and verification of more advanced biometric security protocols such as the L1 biometric protocol.

5. Acknowledgements

The authors would like to thank the Siddaganga Institute of Technology for providing the opportunity to carry out this work.

6. Conflicts of Interest

The authors declare no conflict interest.

7. Authors' Contribution

Pradeep Rajanna: Problem identification, literature review, implementation, model creation, manuscript preparation.
Nagarajaiah Renukamba Sunitha: Research guidance, manuscript correction.

ORCID

Pradeep Rajanna https://orcid.org/0000-0003-4990-1689

Nagarajaiah Renukamba Sunitha https://orcid.org/0000-0003-2865-720X

References

Adeyanju, I. A., Emake, E. D., Olaniyan, O. M., Omidiora, E. O., Adefarati, T., Uzedhe, G. O., & Okomba, N. S. (2021). Digital industrial control systems: Vulnerabilities and security technologies. *Current Applied Science and Technology*, 21(1), 188-207. https://doi.org/10.14456/cast.2021.18

Baillot, P., & Ghyselen, A. (2022). Types for complexity of parallel computation in Pi-calculus. *ACM Transactions on Programming Languages and Systems*, 44(3), 1-50. https://doi.org/10.1145/3495529

Bao, H., Su, Y., Hua, Z., Chen, M., Xu, Q., & Bao, B. (2024). Grid homogeneous coexisting hyperchaos and hardware encryption for 2-D HNN-like map. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 71(9), 4145-4155. https://doi.org/10.1109/TCSI.2024.3423805

Blanchet, B., Cheval, V., & Cortier, V. (2022). ProVerif with Lemmas, induction, fast subsumption, and much more. *IEEE Symposium on Security and Privacy* (69-86). IEEE. https://doi.org/10.1109/SP46214.2022.9833653

Cai, Z., Li, Y., & Zhao, Y. (2024). Murphi2Chisel: A protocol compiler from Murphi to Chisel. *Proceedings of the 15th Asia-Pacific Symposium on Internetware* (pp. 209-218). https://doi.org/10.1145/3671016.3671376

Celi, S., Hoyland, J., Stebila, D., & Wiggers, T. (2022). A tale of two models: Formal verification of KEMTLS via Tamarin. In V. Atluri, R. Di Pietro, C. D. Jensen, & W. Meng

(Eds.). *Computer Security – ESORICS 2022. Lecture Notes in Computer Science. Vol 13556* (pp. 63-83). Springer. https://doi.org/10.1007/978-3-031-17143-7_4

Garanina, N., Staroletov, S., & Gorlatch, S. (2023). *Auto-tuning high-performance programs using model checking in Promela*. https://doi.org/10.48550/arXiv.2305.09130

Jain, A. K., Deb, D., & Engelsma, J. J. (2022). Biometrics: Trust, but verify. *IEEE Transactions on Biometrics, Behavior, and Identity Science*, 4(3), 303-323. https://doi.org/10.1109/TBIOM.2021.3115465

Jiang, G.-J., Li, Z.-Y., Qiao, G., Chen, H.-X., Li, H. Bin, & Sun, H.-H. (2021). Reliability analysis of dynamic fault tree based on binary decision diagrams for explosive vehicle. *Mathematical Problems in Engineering*, 2021, Article 5559475. https://doi.org/10.1155/2021/5559475

Krichen, M. (2023). A survey on formal verification and validation techniques for internet of things. *Applied Sciences*, 13(14), Article 8122. https://doi.org/10.3390/app13148122

Le, T. M. C., Pham, X. T., & Le, V. T. (2024). Advancing security protocol verification: A comparative study of Scyther, Tamarin. *Journal of Technical Education Science*, 19(1), 43-53. https://doi.org/10.54644/jte.2024.1523

Lewis, M., Soudjani, S., & Zuliani, P. (2023). Formal verification of quantum programs: Theory, tools, and challenges. *ACM Transactions on Quantum Computing*, 5(1), 1-35. https://doi.org/10.1145/3624483

Pradeep, R., & Sunitha, N. R. (2022). Formal verification of CHAP PPP authentication protocol for smart city/safe city applications. *Journal of Physics: Conference Series*, 2161, Article 012046. https://doi.org/10.1088/1742-6596/2161/1/012046

Rakotonirina, I., Barthe, G., & Schneidewind, C. (2024). Decision and complexity of Dolev-Yao hyperproperties. *Proceedings of the ACM on Programming Languages*, 8, 1913-1944. https://doi.org/10.1145/3632906

Ram, A., Dutta, M. P., & Chakraborty, S. K. (2024). An authentication mechanism to prevent various security threats in software defined networking by using AVISPA. *Journal of Scientific and Industrial Research*, 83(9), 977-988. https://doi.org/10.56042/jsir.v83i9.6313

Sharma, S., Saini, A., & Chaudhury, S. (2023). A survey on biometric cryptosystems and their applications. *Computers and Security*, 134, Article 103458. https://doi.org/10.1016/j.cose.2023.103458

Sheik, A. T., Maple, C., & Epiphaniou, G. (2022). Considerations for secure MOSIP deployment. *IET Conference Proceedings*, 2022(8), 135-143. https://doi.org/10.1049/icp.2022.2054

Singh, R., & Jackson, S. (2021). Seeing like an Infrastructure: Low-resolution citizens and the Aadhaar identification project. *Proceedings of the ACM on Human-Computer Interaction*, 5(CSCW2), Article 315. https://doi.org/10.1145/3476056

Sireesha, V., & Reddy, S. R. K. (2016). Two levels fusion based multimodal biometric authentication using iris and fingerprint modalities. *International Journal of Intelligent Engineering and Systems*, 9(3), 21-35. https://doi.org/10.22266/ijies2016.0930.03

Thakur, G., Prajapat, S., Kumar, P., & Chen, C.-M. (2024). A privacy-preserving three-factor authentication system for IoT-enabled wireless sensor networks. *Journal of Systems Architecture*, 154, Article 103245. https://doi.org/10.1016/j.sysarc.2024.103245

Yang, K., Swope, A. M., Gu, A., Chalamala, R., Song, P., Yu, S., Godil, S., Prenger, R., & Anandkumar, A. (2023). LeanDojo: Theorem proving with retrieval-augmented language models. *37th Conference on neural information processing systems (NeurIPS 2023)* (pp. 1-40). NeurIPS.