I. J. Computer Network and Information Security, 2023, 3, 15-29
Published Online on June 8, 2023 by MECS Press (http://www.mecs-press.org/)
DOI: 10.5815/ijcnis.2023.03.02

A Modern Mechanism for Formal Analysis of Biometric Authentication Security Protocol

Pradeep R.*
Department of Computer Science and Engineering, Siddaganga Institute of Technology, Tumakuru, 572103, India
E-mail: pradeepr@sit.ac.in
ORCID iD: https://orcid.org/0000-0003-2865-720X
*Corresponding Author

N. R. Sunitha
Department of Computer Science and Engineering, Siddaganga Institute of Technology, Tumakuru, 572103, India
E-mail: nrsunitha@sit.ac.in, nrsunithasit@gmail.com
ORCID iD: https://orcid.org/0000-0003-4990-1689

G. S. Thejas
Department of Computer Science and Electrical Engineering, Tarleton State University, Texas A&M University System, TX, USA
E-mail: sadashiva@tarleton.edu
ORCID iD: https://orcid.org/0000-0001-9606-0128

Received: 07 October 2022; Revised: 10 January 2023; Accepted: 01 February 2023; Published: 08 June 2023

Abstract: A Biometric Authentication Security (BAS) protocol is a method by which a person's unique physiological or behavioral characteristics are used to verify their identity. These characteristics can include fingerprints, facial features, voice patterns, and more. Biometric authentication has become increasingly popular in recent years due to its convenience and perceived security benefits. However, ensuring that the BAS protocols are secure and cannot be easily compromised. Developing a highly secure biometric authentication protocol is challenging, and proving its correctness is another challenge. In this work, we present a modern mechanism for formally analyzing biometric authentication security protocol by taking a Aadhaar Level-0 Iris-based Authentication Protocol as a use case. The mechanism uses formal methods to formally verify the security of the Aadhaar Level-0 Iris-based Authentication protocol, and is based on the widely-used BAN logic (Buruccu, Abadi, and Needham). Using Scyther model checker we analyze the existing biometric authentication protocol and have shown its effectiveness in identifying potential security vulnerabilities. The proposed mechanism is based on a set of security requirements that must be met for the protocol to be considered secure. These requirements include the need for the protocol to be resistant to replay attacks, man-in-the-middle attacks, and impersonation attacks. The mechanism also considers the possibility of an attacker obtaining the biometric data of a legitimate user.

Index Terms: Biometric, Iris, Formal Verification, Model Checking, Scyther.

1. Introduction

India's government started the Aadhaar project, the world's largest national identification project. Its goal is to gather biometric and demographic information about its citizens and store it in a central database. The government has invested at least $890 million in the initiative, and 1036 million customers have signed up. However, several discussions have lately been over the Aadhaar project's privacy and security-related problems [1]. Critical security issues were reported, such as man-in-the-middle attacks, black hole attacks, bogus registration attacks, snooping, etc. In this work, we look at these problems from a security standpoint.

Aadhaar authentication [2] is the procedure by which the Aadhaar number is uploaded online to the Aadhaar system together with additional features, including biometrics, for its verification based on information, data, or documents already present with it. During the authentication transaction, the Aadhaar number is used to look up the resident's record. The demographic and biometric inputs are then compared to the information the resident provided during the enrollment or update process [3]. The potential for biometric data leakage, the dependability of authentication modules, and the lack of transparency in managing biometric data are among the significant security issues in BAS. The application systems for

biometrics are more secure than traditional methods like passwords since each person's unique biometrics must be preserved securely, and reproducing a biometric template is challenging. A biometric data leak may lead to additional security risks since biometrics are unchangeable. The management of biometric data is often handled by a central agent, such as AADHAAR, which guarantees the security and reliability of authentication systems [4]. In order to lessen security risks, current research is entirely concentrated on the secure administration of biometric templates [5]. The specified features are extracted from the data, combined with hash values, and then stored as a template rather than dealing with the real biometric data [6]. Once the template is known, the attackers may employ reverse engineering to approximate the original data. Communication of the biometric data across the network is essential for a biometric-based authentication system, and communication between communication agents must be accomplished safely. The dependability and secrecy of the whole system will be compromised by even the most minor mistake, flaw, or security vulnerability that exists in the authentication protocol [7]. Authenticating the accuracy of the communication protocol is essential for such safety-critical security systems. The usual testing method is insufficient to demonstrate the Aadhaar authentication protocol's reliability and functional correctness. Its many flaws prevent it from proving it [8]. Formal verification [9] is a technique used to mathematically prove the correctness of a system with respect to its specifications. In the context of security protocols, formal verification can be used to ensure that the protocol provides the desired level of security and that it is free of vulnerabilities. One of the most widely used techniques for formal verification of security protocols is symbolic model checking [10]. This technique represents the protocol as a finite state machine and uses automated tools to exhaustively check all possible states and transitions. Model checking method can identify potential security vulnerabilities, such as the ability for an attacker to replay a previous message or intercept a session key. Another technique is using formal verification tools, such as the Scyther [11], Edinburgh Logical Framework (LF), and the pi-calculus. These tools allow for the specification of protocols in a formal language and then use automated theorem provers to prove properties about the protocol, such as authenticity and confidentiality. Formal verification can also be combined with testing to provide a more comprehensive evaluation of a protocol. For example, automated testing can be used to test the protocol implementation against a set of test cases, while formal verification can be used to prove that the implementation is consistent with the formal specification. Using formal verification technique, the reliability and functional correctness of the biometric security protocols can be proved mathematically [12]. In model checking technique an abstract security protocol model will be developed and verified against the security property requirements [13].

For an extensive biometric system such as Aadhaar with 1.4 billion population and ensuring the security of biometric records while storing, accessing, and updating the biometrics is a critical challenge. In a single day, millions of E-KYC transactions will happen in India to verify the user's identity using biometric authentication at facilities such as Banks, PDS shops, Government offices etc. The devices used for E-KYC are either Level-0 or Level-1 security compliance. The L0 devices use the L0 authentication protocol of Aadhaar. The protocols used by Aadhaar system must be free from attacks, errors and security vulnerabilities. In this work, we propose a state-of-the-art mechanism to perform a security analysis of the Aadhaar L0 compliance authentication protocol using formal methodologies and verification procedures. Our work reveals the security vulnerabilities that exist in the Aadhaar L0 Authentication protocol, and we have provided a secure solution by fixing the attacks.

2. Related Works

Several research studies have increased biometric security, from capturing the biometrics to storing and accessing them securely [14]. In [15], the authors utilized the ESTEREL language to formalize the future bus arbitration protocol. They showed how to test, validate, and verify it. Using the model-checking approach, they also showed how to establish mutual exclusion and deadlock-free features formally. In [16], the authors described ways to model distributed concurrent processes or agents that run mainly independently and communicate with one another to coordinate their operations, the formal requirements were represented as LTL, CTL formulas, or using temporal logics. The authors present a technique for automatic reuse of proofs in software verification [17]. The verification of updated codes, mainly code fixes, is guided by proofs about code and proof efforts using Karlsruhe Interactive Verifier (KIV). In [18], the author gives high-level overviews of The formal, interactive, machine-checked software verification area in general and the verification of operating system code in particular. The state of the art at the time, as well as the advantages and disadvantages of machine-checked code proofs, are also covered by the authors. The authors construct a formal model [19] of user processes running simultaneously under a basic operating system that offers inter-process communication and port mapper system calls. They demonstrate a way to do a verification at the code level. The author of [20] shows how formal verification can be used to verify Arrow's theorem through a small project and demonstrates how to encode mathematical proofs in the computer program. In [21], the authors address the challenges in biometric security when biometrics are stored on cloud servers such as Aadhaar and mainly addresses the biometric encryption challenges and the need for biometric encryptions to enhance security. Unsecured communication channels make a replay attack viable. It entails resubmitting biometrics or biometric feature sets that were previously intercepted. This risk may be minimized with the time stamp watermarking technique [22] described in the next section, this risk may be minimized. The brute force attack is another sort of assault that is also viable with password-secured systems. It entails attempting with a huge collection of arbitrary synthetic pictures. This is comparable to making many tries with various terms. In [23], the authors proposed a new clustered minutiae-based scale and rotation invariant fingerprint matching technique [24]. The absence of minutiae

features in the fingerprint regions are the main difficulty encountered in the current latent fingerprint identification method. From the current biometric communication issues, it is clear that the traditional software testing technique is not suitable to detect the design vulnerabilities and achieving security assurance for the security protocols.

3. Formal Analysis of Biometric Authentication Protocol

To the best of our knowledge, we give the first independent examination of the L0 Aadhaar Iris-based authentication models. For instance, we demonstrate an adequate biometric data secret disclosure attack against these methods. With the simplicity of only listening in on several sessions of each protocol between the biometric device and UIDAI CIDR server, one may extract the whole secrets of those protocols using the methods suggested in this article.

For Aadhaar, we suggest a new protocol called Secure L0 Iris Biometric Authentication Protocol (SLIBAP) and provide the findings of our assessment and prototype implementation details. Using GNY logic and the Scyther tool, we officially and informally demonstrate the proposed protocol's security and demonstrate that SLIBAP offers the needed security against various passive and active cryptographic attacks. We have used the Scyther model checker tool (v1.1.3), which is a security protocol formal verification tool that internally uses the Dolev-Yao attacker model to simulate an active attacker. We used Python 2.7, Graphviz-v.6.0.2, and wxPython 2.8 to generate the attack graphs.

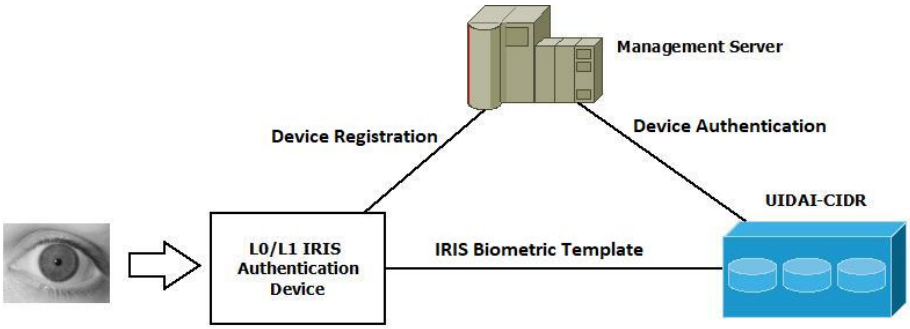


Fig.1. L0 IRIS authentication protocol stakeholders

Fig. 1 shows the stakeholder groups involved in the Aadhaar L0 IRIS Authentication Protocol. When a user's IRIS is captured using an L0 authentication device, the IRIS raw image is captured using the L0 device sensor, and the IRIS template is extracted with the help of the device vendor's Registered Device (RD) service. The management server is used to validate the signature in the device ID idHash. Another role is to do device registration and key rotation. The UIDAI-CIDR is the Aadhaar central biometric repository, where the actual biometric matching happens, and it replies with the authentication status by sending Y or N.

The registered device service offered by device providers that enable collecting and processing biometrics is referred to as the "RD Service." The RD Service sends the caller application an encrypted PID block with signed biometrics that was signed with the device's private key within the registered device's secure zone. Suppose the signature and encryption of biometrics are done inside the software zone at the host OS level. In that case, the device security implementation has level 0 compliance. In this situation, significant consideration must be given to managing private keys to guarantee that they are secured from user or external OS programme access. In level 1 compliance, the biometric data is at the hardware chip level using a pre-stored hardware chip key. The signed and encrypted biometric is called the PID block. The place where a user agent uses or undergoes Aadhaar authentication is called the Authentication User Agency (AUA). Banks, government agencies, and other public and private businesses are examples. To execute secure authentication, all AUAs must be registered with the Aadhaar authentication server, and an Authentication Service Agency (ASA) is a company or other business that connects to UIDAI's data centers via a private, secure network to send authentication requests from multiple AUAs.

3.1. Security Properties Tested

Aliveness: If agent A performs a protocol role under the impression that agent B is present, then agent B has actually fulfilled the function (and is therefore considered to be "alive"). This authentication characteristic is quite unreliable.

Weak agreement: In the case of aliveness, B further thinks that he/she is speaking with A, and as a result, they both "agree" on the agents participating in the protocol.

Agreement: This suggests a lack of agreement, and B plays the part that A usually plays. Finally, A and B agree upon the values in S, such as the computed session key.

Secrecy: Let on the term "role" for secrecy. A computed session key, then it will not become known to the adversary.

Table 1. Notations used in the protocols' description

Symbol	Description
L0-IRIS-RegDev	An L0 Iris-based Authentication Device.
UIDAI-CIDR	Aadhaar UIDAI server.
A	Adversary.
SHA256()	Hash Function.
IIR	Iris Image Record.
DevSerialNum	Iris Device serial number.
DevProviderID	Iris Device Provider ID.
Modelcode	Iris Device Model Code
DevProviderKey	Session Key
TS1	Time Stamp generated by UIDAI.
TS2	Time Stamp generated by L0 Iris Device.
Authentication-Status	Authentication Status from UIDAI.
{data}pk(A)	Encryption of data with public key of role A
Send()	A function/event used by communicating roles to send data over the DY channel
Recv()	A function/event used by communicating roles to receive data over the DY channel
Claim()	A function/event used by communicating roles to prove the security property

We model security processes as sets of roles, with each role comprising several role stages. A role step sends or receives messages that correspond to its message pattern. Before offering an example, we first briefly describe the parts of our specification language. Constants, messages that will be freshly formed (like nonces values), and variables are denoted by the pairwise-disjoint sets Const, Fresh, and Var. We presumptively split the set Var into two sets, AVar and MVar, which stand for agent variables and message variables, respectively. Agent variables are placeholders for the agent names chosen while creating a new role instance, whereas message variables are placeholders for messages (which may also be agent names) received during the execution of a role instance. We refer to the collection of message patterns as the set Ptrn –

$$Ptrn:: = Const \mid Fresh \mid Var \mid h(Ptrn) \mid (Ptrn, Ptrn) \mid \{\mid Ptrn \mid\}_{Ptrn} \mid k_{Ptrn,\ Ptrn} \mid pk_{Ptrn} \mid sk_{Ptrn} \quad (1)$$

A long-term symmetric key shared by a and b is denoted by the pattern ka,b. A's long-term public key is denoted by pka, while a long-term private key is denoted by ska. We designate public-key encryption when k = pka, signing when k = ska, and symmetric encryption in all other cases using the single encryption function { , }k. Notably, we permit the usage of composite messages as symmetric keys. The constructors (,) and h() stand for pairing and hashing, respectively. Let Label consist of several labels. The set RoleStep of roles is what we describe as

$$RoleStep ::= Send_{Label}(Ptrn) \mid Recv_{Label}(Ptrn) \quad (2)$$

Step of a send role Sendl(pt) stands for sending the message that matches the pattern pt. a step receiving role Recvl(pt) indicates that a message matching the pattern pt has been received. The labels serve just to differentiate between transmit (or receive) stages that use the same message pattern and have no practical significance. A role is a finite series R of role steps that is duplicate-free.

$$\forall\ [\![Send]\!]\_1 \in R.\forall v \in FV(pt) \cap MVar. \quad (3)$$

$$\exists l^{\wedge\prime}, pt^{\wedge\prime}.\ [\![Recv]\!]\_(l^{\wedge\prime})\ (pt^{\wedge\prime})\ \_R\ \ [\![Send]\!]\_l\ (pt) \cap v \in FV(pt^{\wedge\prime}) \quad (4)$$

As a result, before a message variable can be delivered in a role, it must first be received. By Role, we represent the collection of all roles. Roles make up a procedure. The collection of all protocols is referred to as Protocol. With a simple challenge-response protocol, we explain how to specify protocols.

3.2. Protocol Execution

The communication agents are allowed to run any number of instances of a protocol P's roles concurrently. Each role instance is referred to as a thread. According to the function they play, threads may create new messages, transmit messages across the network, and receive messages from the network. We presume that an active Dolev-Yao type intrusion has total control over the network. In particular, the intrusion may block and inject communications since it is aware of every message transmitted.

Additionally, the intrusion has gained access to an unlimited number of compromised agents' long-term keys. We provide an operational semantics, described as a state transition system, for protocol execution while the intruder is present. The operational semantics is made up of five components: messages, agent threads, system state, intruder knowledge, and transition system. We go through each of them individually.

Messages: To differentiate between new messages produced by various threads, we utilize their thread IDs. For a thread identifier tid and a message n ∈ Fresh to be freshly generated, to indicate the real fresh message produced by the thread tid for number n, we write n#tid. We use overload notation and write A#TID to represent the set {a#tid | a A; tid ∈ TID} {a#tid ⊣ |a A;tid ∈ TID} for A set.

Given is a set A of agent names unrelated to Const. The set Msg of messages is defined by us.

$$Msg ::= Const \mid Fresh\#TID \mid Agent \mid h(Msg) \mid (Msg, Msg) \mid ⟦\{|Msg|\}⟧\_Msg \mid K\_(Msg, Msg) \mid ⟦Pk⟧\_Msg \mid ⟦Sk⟧\_Msg. \tag{5}$$

Agent threads: The system state keeps track of each thread's role execution as well as the remaining steps for each role.

We represent this data as a partial function.

$$th: TID \rightarrow (Role * RoleStep^*) \tag{6}$$

where *dom(th)* stands for the system's thread IDs. The set *ThreadPool* is defined as the set of all thread pools, and we refer to it as a *thread pool*. Further, the system state contains a variable store $\sigma: Var * TID \rightarrow Msg$ storing for each variable *v* and thread identifier *tid* the contents $\sigma(v, tid)$ assigned to v by thread *tid*. We define the set of all variable stores as $store: Var * TID \rightarrow Msg$. Roles are collections of role steps that describe the messages to be delivered and received as message patterns and are carried out by threads. We shall see later that by considering all potential assignments of messages to a thread's variables, We abstract from the process of instantiating variables while receiving messages. During the execution of a thread *tid* and in the context of a variable store σ, a message pattern *pt* is instantiated to the message $⟦inst⟧\_(\sigma, tid)\ (pt)$ by swapping out all new message patterns for real fresh messages, and all variables for the information that thread *tid* has allocated to them.

$$inst_{\sigma,tid}(pt) = \begin{cases} pt & if\ pt\ is\ a\ Const \\ Pt\ tid & if\ pt\ is\ a\ Fresh \\ \sigma(pt, tid) & if\ pt\ is\ var \\ h\left(inst_{\sigma,tid}(x)\right) & if\ pt = h(x) \\ (inst_{\sigma,tid}(x), inst_{\sigma,tid}(y) & if\ pt = (x, y) \\ \{|inst_{\sigma,tid}(x)|\} inst_{\sigma,tid}(k) & if\ pt = \{|x|\}_k \\ K_{inst_{\sigma,tid}(a), inst_{\sigma,tid}(b)} & if\ pt = k_{a,b} \\ pk_{inst_{\sigma,tid}(a)} & if\ pt = pk_a \\ sk_{inst_{\sigma,tid}(a)} & if\ pt = sk_a \end{cases} \tag{7}$$

System state: The system state maintains track of the thread pool, the values of variables, and a trail documenting which threads carried out which role steps and which messages the intruder learned. This trace establishes a protocol's security characteristics and keeps track of the intruder's knowledge in a system state. A trace is a chain of fundamental occurrences.

$$BasicEvent ::= St(TID;\ RoleStep) \mid Ln(P(Msg)) \tag{8}$$

The primary step event St(tid, s) denotes that the thread tid executed the role step s. Message variables are placeholders for messages (which may also be agent names) received during the execution of a role instance, while agent variables are placeholders for the agent names selected when constructing a new role instance. The terminology used to express the messages exchanged during a protocol session is the basic building block of the protocol model. The union of the set of global constants Const, the set of terms newly formed during protocol execution Nonce, and the set of variables Var is what we refer to as the "basic terms set." The group of words Term comprises the fundamental terms and any term created using one of the two constructors, tupling (p,p') and encryption {|p|}p' of a term. The list of thread identifiers is indicated by the set ID, which is defined below.

$$BasicTerm ::= Const \mid Nonce\ ID \mid Var\ ID \tag{9}$$

$$Term ::= BasicTerm \mid (Term; Term) \mid \{|Term|\} Term \tag{10}$$

Execution model: The set Trace is defined as $(TID \times Event)^*$ and represents potential execution histories. The four-tuple $(tr, IK, th, \sigma\, Test) \in Trace \times P(Term) \times (TID \to\times Event*) \times Sub$ that represents the state of our system consists of a trace (*tr*), the adversary's knowledge (*IK*), a partial function (*th*) that maps thread identifiers of initiated threads to sequences of events (th), and (4) the role to agent and variable assignments of the test thread. We incorporate the trace as part of the state and correct the test substitution at system startup to make it easier to define the partner function afterward.

TestSub$_P$. We define the set of test substitutes for a protocol P. *TestSubP* is the collection of ground replacements for Test such that $dom(\sigma_{Test}) = dom(P) \cup \{v\#Test \mid v \in Var\}\, and\, \forall r \in dom(P).\ \sigma_{Test}(r) \in Agent.$

The initial adversary knowledge, or AK_0, is defined as - $AK_0 = Agent \cup \{pk(a) \mid a \in Agent\} \cup \{c\#tid_A \mid c \in Fresh\}$

The adversary initially has access to each agent's name, public key, and a list of constants it created. We represent these constants with new symbols that are connected to the adversary's thread identification, tid_A. The initial opponent information does not include any long-term secret keys, in contrast to most Dolev-Yao models. The enemy might discover these via long-term key reveal (LKR) occurrences.

The collection of initial system states IS(P) for protocol P is defined as

$$IS(P) = \sum_{\sigma_{Test} \in TestSub_P}^{\sigma_{Test}} \{(<>, AK_0, \emptyset,\ \sigma_{Test}\ ). \tag{11}$$

A transition system combining adversary rules from equation 13 with the execution rules from equation 12 defines the semantics of a protocol P. The execution guidelines are presented first. A role R of the protocol P is started new (as a thread) by the createP rule. The thread is given a brand-new thread identification tid to set it apart from other threads, the adversary thread and the test thread. The protocol *P* is a parameter for the rule. The replacement changes the role names of P, which might appear in events related to the role, into agent names. The test thread is also started by the *createTest*$_P$ rule. However, it accepts an extra argument R_{Test}, which stands in for the test role and will be instantiated in the specification of the transition relation, rather than selecting a random role. Additionally, the σ_{Test} Test is utilised rather of picking a random.

A message m is sent to the network via the send rule. The receive rule, in contrast, only acknowledges network messages that match the pattern *pt*, where *pt* is a word that could include free variables. The remaining procedure stages *l* are then modified to reflect the resultant substitution σ. Note that we have associated the network with the attacker, as is customary. As a result, communications are delivered to and received from *IK* directly. The last three rules support the rules that our attacker will use against us. The session keys rule identifies a group of words as session keys, the state rule identifies the current local state, and the produce rule identifies newly created terms.

$$\frac{R \in dom(P)\ \sigma \in Role \to Agent\ tid \notin (dom(th) \cup \{tid_A, Test\})\ \ l = thread(P(R), tid, \sigma)}{(tr, IK, th, \sigma_{Test}) \to \left(tr\hat{}<\left(tid, create\left(R, \sigma(R)\right)\right)>, IK, th[l \leftarrow tid], \sigma_{Test}\right)}\ [create_P]$$

$$\frac{a = \sigma_{Test}(R_{Test})\ Test \notin dom(th) \quad l = thread(P(R_{Test}), Test, \sigma_{Test}))}{(tr, IK, th, \sigma_{Test}) \to (tr\hat{}<(Test, create(R_{Test}, a))>, IK, th[l \leftarrow Test], \sigma_{Test})}\ [createTest_P]$$

$$\frac{th(tid) = <send(m)>\hat{}\, l}{(tr, IK, th, \sigma_{Test}) \to (tr\hat{}<(tid, send(m))>, IK \cup \{m\}, th[l \leftarrow tid], \sigma_{Test})}\ [send]$$

$$\frac{th(tid) = <recv(pt)>\hat{}\, l \quad IK \vdash \sigma(pt) \quad dom(\sigma) = FV(pt)}{(tr, IK, th, \sigma_{Test}) \to (tr\hat{}<(tid, recv(\sigma(pt)))>, IK, th[\sigma(l) \leftarrow tid], \sigma_{Test})}\ [recv] \tag{12}$$

$$\frac{th(tid) = <generate(M)>\hat{}\, l}{(tr, IK, th, \sigma_{Test}) \to (tr\hat{}<(tid, generate(M))>, IK, th[l \leftarrow tid], \sigma_{Test})}\ [generate]$$

$$\frac{th(tid) = <state(M)>\hat{}\, l}{(tr, IK, th, \sigma_{Test}) \to (tr\hat{}<(tid, state(M))>, IK, th[l \leftarrow tid], \sigma_{Test})}\ [state]$$

$$\frac{th(tid) = <sessionkeys(M)>\hat{}\, l}{(tr, IK, th, \sigma_{Test}) \to (tr\hat{}<(tid, sessionkeys(M))>, IK, th[l \leftarrow tid], \sigma_{Test})}\ [sessionkeys]$$

$$\frac{a \in Agent\ a\ 6 \in \{\sigma_{Test}(R) \mid R \in dom(P)\}}{(tr, IK, th, \sigma_{Test}) \to (tr\hat{}<(tid_A, LKR(a))>, IK \cup LongTermKeys(a), th, \sigma_{Test})}\ [LKR_{others}]$$

$$\frac{a = \sigma_{Test}(R_{Test}) \qquad a \notin \{\sigma_{Test}(R) \mid R \in dom(P) \setminus \{R_{Test}\}\}}{(tr, IK, th, \sigma_{Test}) \to (tr\hat{}<(tid_A, LKR(a))>, IK \cup LongTermKeys(a), th, \sigma_{Test})}\ [LKR_{actor}]$$

$$\frac{a \in Agent \qquad th(Test) = <>}{(tr, IK, th, \sigma_{Test}) \to (tr\hat{}<(tid_A, LKR(a))>, IK \cup LongTermKeys(a), th, \sigma_{Test})}\ [LKR_{after}]$$

$$\frac{a \in Agent \quad th(Test) = <> \quad tid \in Partner(tr, \sigma_{Test}) \quad th(Test) = <>}{(tr, IK, th, \sigma_{Test}) \rightarrow (tr^{\frown} <(tid_A, LKR(a))>, IK \cup LongTermKeys(a), th, \sigma_{Test})} [LKR_{aftercorrect}] \quad (13)$$

$$\frac{tid \neq Test \quad tid \notin Partner(tr, \sigma_{Test})}{(tr, IK, th, \sigma_{Test}) \rightarrow (tr^{\frown} <(tid_A, SKR(tid))>, IK \cup union((tr \downarrow tid) \downarrow sessionkeys), th, \sigma_{Test})} [SKR]$$

$$\frac{tid \neq Test \quad tid \notin Partner(tr, \sigma_{Test}) \quad th(Test) \neq <>}{(tr, IK, th, \sigma_{Test}) \rightarrow (tr^{\frown} <(tid_A, SR(tid))>, IK \cup last((tr \downarrow tid) \downarrow state), th, \sigma_{Test})} [SR]$$

$$\frac{}{(tr, IK, th, \sigma_{Test}) \rightarrow (tr^{\frown} <(tid_A, RNR(tid))>, IK \cup union((tr \downarrow tid) \downarrow generate), th, \sigma_{Test})} [RNR]$$

We divide local data into three categories: session keys, randomness, and additional types like the outcomes of intermediate values. The session-key reveal event SKR(tid), and state reveal event SR(tid) in the adversary-compromise paradigm indicate that the adversary has access to the session key or the local state of the thread tid. The session keys and state events, respectively, demonstrate this. The state's contents are erased when the thread ends since they change over time. The last state marker for the state contents and the third premise, which specifies that the thread tid has not ended, represent this in the SR rule. The attacker is depicted as knowing the random numbers created in the thread tid via the random number disclosure event RNR (tid).

3.3. Existing L0 IRIS Authentication Protocol

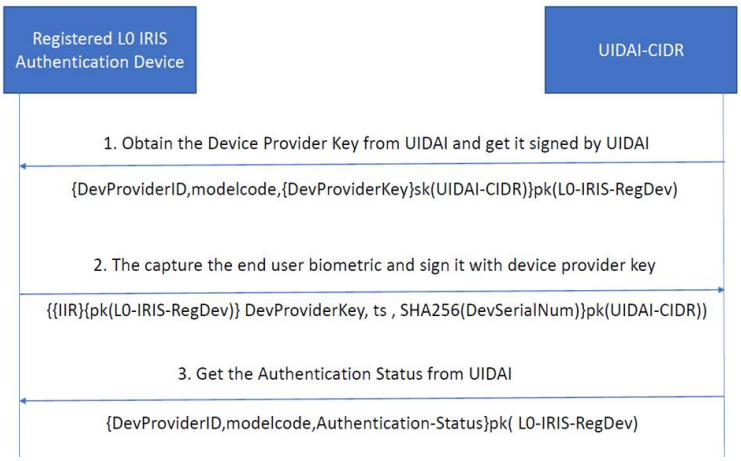


Fig.2. Existing UIDAI L0 IRIS authentication protocol steps

The L0 Aadhaar IRIS authentication protocol mainly involves 3 steps, as shown in Figure 2. In Step 1, The device provider obtains a device provider key signed by UIDAI.

BasicTerm::= SessionKey | string| SHA256()| ImageRecord| Nonce $_{ID}$ | Binary | timestamp | DevSerialNum| IIR| DevProviderID| Modelcode| DevProviderKey| ts|ts1|ts2| Authentication-Status

Term ::= BasicTerm | (Term;Term) | {|Term|}$_{Term}$

E $_{pk(L0\text{-}IRIS\text{-}RegDev)}$ (DevProviderID||modelcode||Sign $_{sk(UIDAI\text{-}CIDR)}$ (DevProviderKey))

In Step 2, the users' Iris biometrics are captured using RD service, and the Iris Image Record IIR is created. Finally, the PID block is created and signed with the device provider key. The PID block contains the Iris authentication device signed biometric record, a time stamp, and device's unique serial number generated using SHA256.

Biometric Record BR=Sign $_{pk(L0\text{-}IRIS\text{-}RegDev)}$ (IIR)

$E_{pk(UIDAI\text{-}CIDR)}$ ($E_{DevProviderKey}$ ((BR)|| ts ||SHA256(DevSerialNum))

In Step 3, the authentication happens at UIDAI-CIDR, and it will send the authentication status by sending Y or N in the field of Authentication status. In this work, the abstract security protocol model of the Aadhaar L0 single IRIS authentication protocol is developed from its standard authentication protocol specification document [2]. The protocol model has two communicating agents or roles, namely RegDevL0-IRIS and UIDAI-CIDR. These roles communicate data over the Dolev-Yao adversary channel, which is the strongest active adversary.

Model 1- Existing Aadhaar L0 IRIS Authentication Protocol model.

```
usertype SessionKey;
usertype string;
hashfunction SHA256;
usertype ImageRecord;
usertype Binary;
usertype timestamp;
protocol Aadhaar(L0-IRIS-RegDev,UIDAI-CIDR)
{
role L0-IRIS-RegDev
{
fresh DevSerialNum: string;
fresh IIR: ImageRecord;
var DevProviderID: string;
var modelcode: string;
var DevProviderKey: SessionKey;
fresh ts: timestamp;
var ts1: Nonce;
fresh ts2: Nonce;
var Authentication-Status:Binary;
recv_1(UIDAI-CIDR, L0-IRIS-RegDev,{DevProviderID, modelcode, {DevProviderKey}sk(UIDAI-CIDR)}pk(L0-IRIS-RegDev));
send_2(L0-IRIS-RegDev, UIDAI-CIDR,{{IIR}{pk(L0-IRIS-RegDev)}DevProviderKey, ts, SHA256(DevSerialNum)}pk(UIDAI-CIDR));
recv_3(UIDAI-CIDR, L0-IRIS-RegDev, {DevProviderID, modelcode, Authentication-Status}pk(L0-IRIS-RegDev));
claim_c1(L0-IRIS-RegDev,Secret,ts);
claim_c2(L0-IRIS-RegDev,Secret,IIR);
claim_c3(L0-IRIS-RegDev,Secret,DevSerialNum);
claim_c4(L0-IRIS-RegDev,Secret,Authentication-Status);
claim_c5(L0-IRIS-RegDev,Secret,DevProviderKey);
claim_c6(L0-IRIS-RegDev,Secret,modelcode);
claim_c7(L0-IRIS-RegDev,Secret,DevProviderID);
claim_c8(L0-IRIS-RegDev,Secret,sk(UIDAI-CIDR));
claim_c9(L0-IRIS-RegDev,Alive);
claim_c10(L0-IRIS-RegDev,Weakagree);
claim_c11(L0-IRIS-RegDev,Nisynch);
claim_c12(L0-IRIS-RegDev,Niagree);
}
role UIDAI-CIDR
{
fresh DevProviderID: string;
fresh modelcode: string;
fresh DevProviderKey: SessionKey;
var DevSerialNum: string;
var IIR: ImageRecord;
var ts: timestamp;
fresh ts1: Nonce;
send_1(UIDAI-CIDR, L0-IRIS-RegDev, {DevProviderID, modelcode, {DevProviderKey}sk(UIDAI-CIDR)}pk(L0-IRIS-RegDev));
claim(UIDAI-CIDR,Secret,DevProviderKey);
var ts2: Nonce;
recv_2(L0-IRIS-RegDev, UIDAI-CIDR, {{IIR}{pk(L0-IRIS-RegDev)}DevProviderKey, ts, SHA256(DevSerialNum)}pk(UIDAI-CIDR));
fresh Authentication-Status:Binary;
send_3(UIDAI-CIDR, L0-IRIS-RegDev, {DevProviderID, modelcode, Authentication-Status}pk( L0-IRIS-RegDev));
claim_u1(UIDAI-CIDR,Secret,DevProviderKey);
claim_u2(UIDAI-CIDR,Secret,DevSerialNum);
claim_u3(UIDAI-CIDR,Secret,Authentication-Status);
claim_u4(UIDAI-CIDR,Secret,DevProviderID);
claim_u5(UIDAI-CIDR,Secret,sk(UIDAI-CIDR));
claim_u6(UIDAI-CIDR,Secret,Authentication-Status);
claim_u7(UIDAI-CIDR,Secret,IIR);
```

claim_u8(UIDAI-CIDR,Secret,ts);
claim_u9(UIDAI-CIDR,Nisynch);
claim_u10(UIDAI-CIDR,Niagree);
claim_u11(UIDAI-CIDR,Alive);
}
}

Model 1 shows the existing L0 IRIS authentication protocol abstract security protocol model developed in the SPDL language (Security Protocol Description Language). RegDevL0-IRIS and UIDAI-CIDR are the two communicating roles in the model. The registered L0 IRIS authentication device is represented by the role RegDevL0-IRIS. The size of public, private, and session keys used in the model is 2048 bit. Authentication involves three main steps. In step 1, the RegDevL0-IRIS obtains a device provider key that is signed by UIDAI-CIDR in a send_1 event where the DevProviderKey value is signed by the secret key of UIDAI-CIDR. Then, along with it, DevProviderID and model code data are finally encrypted using the public key of the registered L0 IRIS device as follows: {|DevProviderID, modelcode, {DevProviderKey}sk(UIDAI-CIDR)|}pk(L0-IRIS-RegDev)

The encrypted send_1 data is decrypted by RegDevL0-IRIS in the recv_1 event and obtained by the DevProviderKey. In step 2, the PID block is created in which the biometric data (IRIS Image Record, IIR) are captured and signed using the registered L0 device. Finally, an encrypted PID block is created, a biometric capture time stamp is generated, and a unique device serial number is generated using the SHA256 hash function and sent to UIDAI-CIDR for authentication. The device serial number remains constant throughout the lifecycle of the L0 device.

{|{IIR}pk(RegDevL0)}DevProviderKey, {DevSerialNum, base64(DevSerialNum), ts, SHA256(DevSerialNum)|}pk(UIDAI-CIDR)

In the recv_2 event, role UIDAI-CIDR decrypts the encrypted send_2 data and securely obtains the biometric data. In the final step, the UIDAI-CIDR responds back to the authentication status by replying Y or N based on the match in the send_3 event, and it is decrypted by role RegDevL0-IRIS in the recv_3 event.

{|DevProviderID, modelcode,Authentication-Status|}pk(L0-IRIS-RegDev)

The main goal of the Aadhaar L0 IRIS device authentication protocol is to ensure biometric data security and confidentiality, i.e., the confidentiality of IIR biometric data when transmitted over an untrusted channel. Figure 4 shows the verification result of the existing Aadhaar L0 IRIS Authentication model result, in which the model fails to satisfy the c4, c5, c6, c7, c10, c11, and c12 security claims by the L0-IRIS-RegDev role.

The claim_c4, c5, c6, and c7 are the secrecy security claims claimed by the L0-IRIS-RegDev role. The existing Iris authentication protocol fails to provide the secrecy of the Authentication-Status, DevProviderKey, modelcode, and DevProviderID data when sent over the untrusted channel. Also, it fails to satisfy Weakagree, Nisynch, and Niagree security properties. For every claim that failed, the corresponding attacks and attack graphs were generated using the Scythe tool. The UIDAI-CIDR role satisfies all the security claims from u1 to u11 mentioned in the role description.

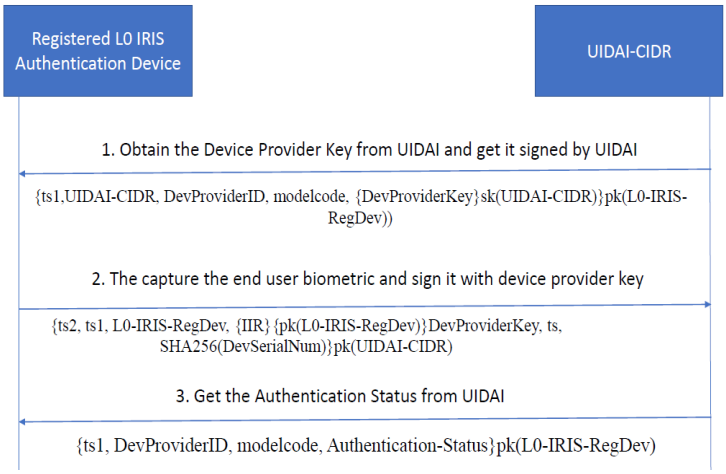


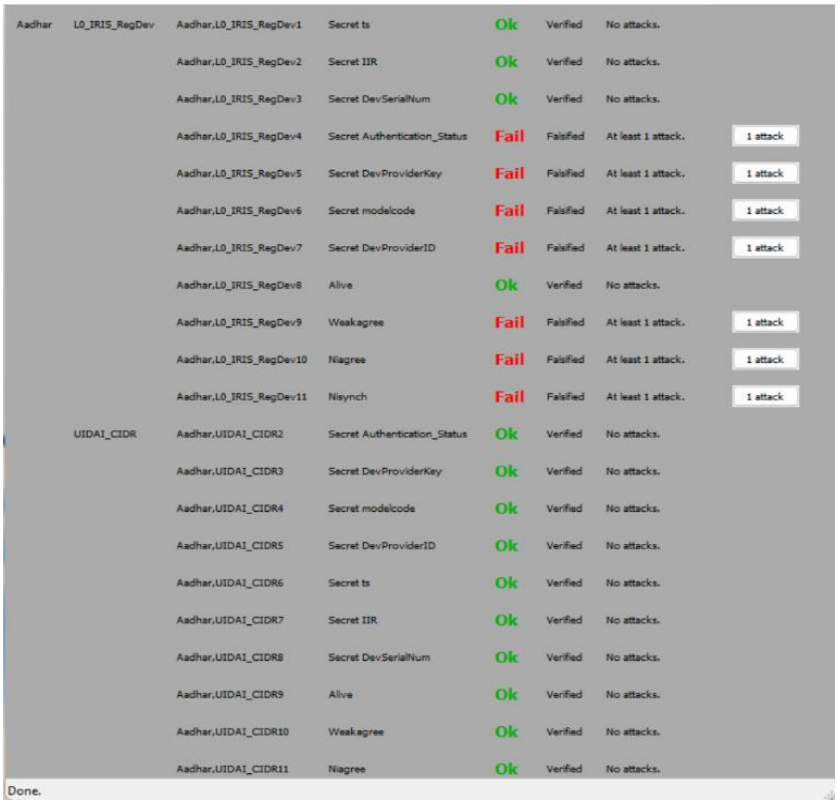
Fig.3. Proposed Secure L0 IRIS Authentication protocol steps

3.4. Proposed L0 IRIS Authentication Protocol

Model 2 Proposed Secure Aadhaar L0 IRIS Authentication Protocol model.
usertype SessionKey;

```
usertype string;
hashfunction SHA256;
usertype ImageRecord;
usertype MinutiaeRecord;
usertype Binary;
const DSA: Function;
const base64: Function;
usertype timestamp;
protocol Aadhaar(L0-IRIS-RegDev,UIDAI-CIDR)
{
role L0-IRIS-RegDev
{
fresh Cik:Pre-CertifiedHardwarechipKey;
fresh DevSerialNum: string;
fresh IIR: ImageRecord;
var DevProviderID: string;
var modelcode: string;
var DevProviderKey: SessionKey;
fresh ts: timestamp;
var ts1: Nonce;
fresh ts2: Nonce;
var Authentication-Status:Binary;
recv_1(UIDAI-CIDR,L0-IRIS-RegDev,{ts1,UIDAI-CIDR, DevProviderID, modelcode, {DevProviderKey}sk(UIDAI-
CIDR)}pk(L0-IRIS-RegDev));
send_2(L0-IRIS-RegDev,UIDAI-CIDR,{ts2, ts1, L0-IRIS-RegDev, {IIR}{pk(L0-IRIS-RegDev)}DevProviderKey, ts,
SHA256(DevSerialNum)}pk(UIDAI-CIDR));
recv_3(UIDAI-CIDR, L0-IRIS-RegDev, {ts1, DevProviderID, modelcode, Authentication-Status}pk(L0-IRIS-RegDev));
claim_c1(L0-IRIS-RegDev,Secret,ts);
claim_c2(L0-IRIS-RegDev,Secret,IIR);
claim_c3(L0-IRIS-RegDev,Secret,DevSerialNum);
claim_c4(L0-IRIS-RegDev,Secret,Authentication-Status);
claim_c5(L0-IRIS-RegDev,Secret,DevProviderKey);
claim_c6(L0-IRIS-RegDev,Secret,modelcode);
claim_c7(L0-IRIS-RegDev,Secret,DevProviderID);
claim_c8(L0-IRIS-RegDev,Secret,sk(UIDAI-CIDR));
claim_c9(L0-IRIS-RegDev,Alive);
claim_c10(L0-IRIS-RegDev,Weakagree);
claim_c11(L0-IRIS-RegDev,Nisynch);
claim_c12(L0-IRIS-RegDev,Niagree);
}
role UIDAI-CIDR
{
fresh DevProviderID: string;
fresh modelcode: string;
fresh DevProviderKey: SessionKey;
var DevSerialNum: string;
var IIR: ImageRecord;
var ts: timestamp;
fresh ts1: Nonce;
fresh Authentication-Status:Binary;
var ts2: Nonce;
send_1(UIDAI-CIDR, L0-IRIS-RegDev,{ts1, UIDAI-CIDR, DevProviderID, modelcode, {DevProviderKey}sk(UIDAI-
CIDR)}pk(L0-IRIS-RegDev));
recv_2(L0-IRIS-RegDev, UIDAI-CIDR,{ts2, ts1, L0-IRIS-RegDev, {IIR}{pk(L0-IRIS-RegDev)}DevProviderKey, ts,
SHA256(DevSerialNum)}pk(UIDAI-CIDR));
send_3(UIDAI-CIDR,L0-IRIS-RegDev,{ts2, DevProviderID, modelcode, Authentication-Status}pk( L0-IRIS-RegDev));
claim_u1(UIDAI-CIDR,Secret,DevProviderKey);
claim_u2(UIDAI-CIDR,Secret,DevSerialNum);
claim_u3(UIDAI-CIDR,Secret,Authentication-Status);
claim_u4(UIDAI-CIDR,Secret,DevProviderID);
claim_u5(UIDAI-CIDR,Secret,sk(UIDAI-CIDR));
claim_u6(UIDAI-CIDR,Secret,Authentication-Status);
```

```
claim_u7(UIDAI-CIDR,Secret,IIR);
claim_u8(UIDAI-CIDR,Secret,ts);
claim_u9(UIDAI-CIDR,Nisynch);
claim_u10(UIDAI-CIDR,Niagree);
claim_u11(UIDAI-CIDR,Alive);
}
}
```



Aadhar	L0_IRIS_RegDev	Aadhar,L0_IRIS_RegDev1	Secret ts	Ok	Verified	No attacks.	
		Aadhar,L0_IRIS_RegDev2	Secret IIR	Ok	Verified	No attacks.	
		Aadhar,L0_IRIS_RegDev3	Secret DevSerialNum	Ok	Verified	No attacks.	
		Aadhar,L0_IRIS_RegDev4	Secret Authentication_Status	Fail	Falsified	At least 1 attack.	1 attack
		Aadhar,L0_IRIS_RegDev5	Secret DevProviderKey	Fail	Falsified	At least 1 attack.	1 attack
		Aadhar,L0_IRIS_RegDev6	Secret modelcode	Fail	Falsified	At least 1 attack.	1 attack
		Aadhar,L0_IRIS_RegDev7	Secret DevProviderID	Fail	Falsified	At least 1 attack.	1 attack
		Aadhar,L0_IRIS_RegDev8	Alive	Ok	Verified	No attacks.	
		Aadhar,L0_IRIS_RegDev9	Weakagree	Fail	Falsified	At least 1 attack.	1 attack
		Aadhar,L0_IRIS_RegDev10	Niagree	Fail	Falsified	At least 1 attack.	1 attack
		Aadhar,L0_IRIS_RegDev11	Nisynch	Fail	Falsified	At least 1 attack.	1 attack
	UIDAI_CIDR	Aadhar,UIDAI_CIDR2	Secret Authentication_Status	Ok	Verified	No attacks.	
		Aadhar,UIDAI_CIDR3	Secret DevProviderKey	Ok	Verified	No attacks.	
		Aadhar,UIDAI_CIDR4	Secret modelcode	Ok	Verified	No attacks.	
		Aadhar,UIDAI_CIDR5	Secret DevProviderID	Ok	Verified	No attacks.	
		Aadhar,UIDAI_CIDR6	Secret ts	Ok	Verified	No attacks.	
		Aadhar,UIDAI_CIDR7	Secret IIR	Ok	Verified	No attacks.	
		Aadhar,UIDAI_CIDR8	Secret DevSerialNum	Ok	Verified	No attacks.	
		Aadhar,UIDAI_CIDR9	Alive	Ok	Verified	No attacks.	
		Aadhar,UIDAI_CIDR10	Weakagree	Ok	Verified	No attacks.	
		Aadhar,UIDAI_CIDR11	Niagree	Ok	Verified	No attacks.	

Done.

Fig.4. Existing L0 IRIS Authentication protocol result

Model 2 is the proposed L0 IRIS authentication protocol, an abstract security protocol model developed in the SPDL language. The model has two communicating roles, namely L0-IRIS-RegDev and UIDAI-CIDR. The registered L0 IRIS authentication device is represented by the role L0-IRIS-RegDev. Authentication involves three main steps. In step 1, the L0-IRIS-RegDev obtains a device provider key that is signed by UIDAI-CIDR in a send_1 event where the DevProviderKey value is signed by the secret key of UIDAI-CIDR. Then along with it, time stamp TS1, UIDAI-CIDR identity, DevProviderID, and model code data are finally encrypted using the public key of the registered L0 IRIS device as follows:

{| ts1, UIDAI-CIDR, DevProviderID, modelcode, {DevProviderKey}sk(UIDAI-CIDR)|}pk(L0-IRIS-RegDev)

The encrypted send_1 data is decrypted by RegDevL0-IRIS in the recv_1 event and obtained by the DevProviderKey. In step 2, the biometric data (IRIS Image Record, IIR) are captured using the L0 RD service and signed using the registered L0 device public key pk (L0-IRIS-RegDev). Finally, an encrypted PID block is created, a biometric capture time stamp ts is generated, and a unique device serial number is generated using the SHA256 hash function and sent to UIDAI-CIDR for authentication. The device serial number remains constant throughout the lifecycle of the L0 device.

{| ts2, ts1, L0-IRIS-RegDev, {IIR}{pk(L0-IRIS-RegDev)}DevProviderKey, ts,
SHA256(DevSerialNum)|}pk(UIDAI-CIDR))

In the recv_2 event, role UIDAI-CIDR decrypts the encrypted send_2 data and securely obtains the biometric data. In the final step, the UIDAI-CIDR responds back to the authentication status by replying Y or N based on the match in the send_3 event, and it is decrypted by role RegDevL0-IRIS in the recv_3 event.

{| ts2, DevProviderID, modelcode, Authentication-Status |}pk(L0-IRIS-RegDev))

The main goal of the Aadhaar L0 IRIS device authentication protocol is to ensure biometric data security and confidentiality, i.e., the confidentiality of IIR biometric data when transmitted over an untrusted channel. Figure 5. Shows the verification result of the proposed Aadhaar L0 IRIS Authentication protocol model result, in which the model of all the claims made by both the roles.

Fig.5. Proposed L0 IRIS Authentication protocol result

Table 2. Verification Time consumed by L0 existing and proposed system

Number of Rounds	Verification Times in Seconds	
	L0 Existing	L0 Proposed
1	1	1
5	8	8
10	15	20
15	25	30
20	35	45
25	50	65

4. Results and Discussion

Figures 4 and 5 show the verification results of the existing and proposed Aadhaar L0 IRIS Authentication protocol models, respectively, in which the existing model fails to satisfy the c4, c5, c6, c7, c10, c11, and c12 security claims of the L0-IRIS-RegDev role. The claim_c4, c5, c6, and c7 are the secrecy security claims claimed by the L0-IRIS-RegDev role. When sent over the untrusted channel, the existing Iris authentication protocol fails to provide the secrecy of the Authentication-Status, DevProviderKey, modelcode, and DevProviderID data. Also, it fails to satisfy Weakagree, Nisynch, and Niagree security properties. For every failed claim, corresponding attacks and graphs were generated using the Scythe tool. The UIDAI-CIDR role satisfies all the security claims from u1 to u11 mentioned in the role description. All these attacks were fixed in the proposed L0 authentication protocol model using additional nonce values and role identities. Figure 5 shows the verification results of the proposed L0 Iris authentication models, which is highly secure and free from cryptographic attacks. Figure 6 shows the verification time consumed vs. the number of verifications rounds in which the L0 existing system model takes the least time compared to the proposed L0 model. This is because as the number of variables increases, the number of states of the verification model increases exponentially. Similarly, the L0

existing system takes comparatively less verification time than the L0 proposed system because the proposed L0 protocol model uses additional nonce values.

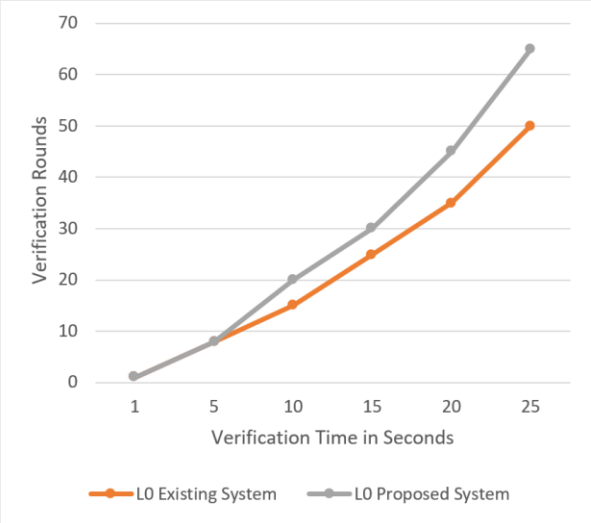


Fig.6. L0 Existing and proposed model verification time comparison

5. Conclusions

As this study shows, developing safe cryptographic methods is very challenging. For the Aadhaar context, we analyzed the existing L0 Iris-based authentication protocol. Model checking the existing L0 Iris authentication protocol model reveals the presence of cryptographic attacks; these attacks were fixed using our proposed work by including additional nonce values and identities of the communicating roles. We suggested a trustworthy and secure user authentication protocol. The proposed L0 Iris authentication protocol models provide confidentiality for the Iris biometric data when communicated over an insecure channel. By comparing L0 existing and proposed model verification time, it is clear that as the number of data variables increases in the protocol model, the verification time consumed by model checker also increases exponentially. The proposed model takes 13% more verification time than the existing system model. In conclusion, formal verification is a powerful technique that can mathematically prove a security protocol's correctness. Providing a high degree of confidence in the protocol's security can be a valuable tool for identifying potential vulnerabilities and ensuring the protection of sensitive information. However, it should be used in conjunction with other techniques to provide a comprehensive evaluation of the protocol's security.

References

[1] I. Pali, L. Krishania, D. Chadha, A. Kandar, G. Varshney, and S. Shukla, “A Comprehensive Survey of Aadhar and Security Issues,” 2020, [Online]. Available: http://arxiv.org/abs/2007.09409

[2] B. C. Conference, “Biometric Authentication Features of UID (Aadhaar) Authentication,” no. September, 2013.

[3] P. Patil and S. Jagtap, “Multi-modal biometric system using finger knuckle image and retina image with template security using PolyU and DRIVE database,” *Int. J. Inf. Technol.*, vol. 12, no. 4, pp. 1043–1050, 2020, doi: 10.1007/s41870-020-00501-0.

[4] T. J. Tsitaitse, Y. Cai, S. L. Suntu, and M. N. U. khan, “Secure and Fast Handovers Authentication Methods for Wi-Fi Based Networks: A Review perspective,” *Int. J. Comput. Networks Appl.*, vol. 5, no. 3, pp. 32–42, 2018, doi: 10.22247/ijcna/2018/49434.

[5] K. Nandakumar, A. K. Jain, and A. Nagar, “Biometric template security,” *EURASIP J. Adv. Signal Process.*, vol. 2008, 2008, doi: 10.1155/2008/579416.

[6] A. Dantcheva, P. Elia, and A. Ross, “What else does your biometric data reveal? A survey on soft biometrics,” *IEEE Trans. Inf. Forensics Secur.*, vol. 11, no. 3, pp. 441–467, 2016, doi: 10.1109/TIFS.2015.2480381.

[7] N. Asokan, V. Niemi, and K. Nyberg, “Man-in-the-middle in tunnelled authentication protocols,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 3364 LNCS, pp. 28–41, 2005, doi: 10.1007/11542322_6.

[8] M. Tanveer, A. U. Khan, H. Shah, A. Alkhayyat, S. A. Chaudhry, and M. Ahmad, “ARAP-SG: Anonymous and Reliable Authentication Protocol for Smart Grids,” *IEEE Access*, vol. 9, pp. 143366–143377, 2021, doi: 10.1109/ACCESS.2021.3121291.

[9] Rekha and R. Gupta, “Elliptic curve cryptography based secure image transmission in clustered wireless sensor networks,” *Int. J. Comput. Networks Appl.*, vol. 8, no. 1, pp. 67–78, 2021, doi: 10.22247/IJCNA/2021/207983.

[10] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang, “Symbolic model checking: 1020 States and beyond,” *Inf. Comput.*, vol. 98, no. 2, pp. 142–170, 1992, doi: 10.1016/0890-5401(92)90017-A.

[11] H. Foster, *Assertion-based verification: Industry myths to realities*. 1973.

[12] K. Hofer-Schmitz and B. Stojanović, “Towards formal verification of IoT protocols: A Review,” *Comput. Networks*, vol. 174, no. March, 2020, doi: 10.1016/j.comnet.2020.107233.

[13] S. Zhang, A. Burns, J. Chen, and E. S. Lee, “Hard real-time communication with the timed token protocol: Current state and

challenging problems," *Real-Time Syst.*, vol. 27, no. 3, pp. 271–295, 2004, doi: 10.1023/B:TIME.0000029051.60313.06.
[14] S. Liu and M. Silverman, "Practical guide to biometric security technology," *IT Prof.*, vol. 3, no. 1, pp. 27–32, 2001, doi: 10.1109/6294.899930.
[15] F. Boussinot, S. Ramesh, R. K. Shyamasundar, and R. De Simone, "Validation and analysis of the futurebus arbitration protocol: A case study," *Sadhana*, vol. 21, no. 2, pp. 185–211, 1996, doi: 10.1007/BF02745519.
[16] K. Lodaya, M. Mukund, R. Ramanujam, and P. S. Thiagarajan, "Models and logics for true concurrency," *Sadhana*, vol. 17, no. 1, pp. 131–165, 1992, doi: 10.1007/BF02811341.
[17] G. Klein, R. Huuck, and B. Schlich, "Operating system verification," *J. Autom. Reason.*, vol. 42, no. 2–4, pp. 123–124, 2009, doi: 10.1007/s10817-009-9126-9.
[18] W. R. Bevier, W. A. Hunt, J. S. Moore, and W. D. Young, "An approach to systems verification," *J. Autom. Reason.*, vol. 5, no. 4, pp. 411–428, 1989, doi: 10.1007/BF00243131.
[19] F. Flammini, A. Lazzaro, and N. Mazzocca, "Modeling of railway logics for reverse enginering, verification and refactoring," *Int. J. Saf. Secur. Eng.*, vol. 1, no. 1, pp. 77–94, 2011, doi: 10.2495/SAFE-V1-N1-77-94.
[20] M. J. Scott and E. K. Antonsson, "Arrow's Theorem and engineering design decision making," *Res. Eng. Des. - Theory, Appl. Concurr. Eng.*, vol. 11, no. 4, pp. 218–228, 1999, doi: 10.1007/s001630050016.
[21] V. Kakkad, M. Patel, and M. Shah, "Biometric authentication and image encryption for image security in cloud framework," *Multiscale Multidiscip. Model. Exp. Des.*, vol. 2, no. 4, pp. 233–248, 2019, doi: 10.1007/s41939-019-00049-y.
[22] M. Ganavi, S. Prabhudeva, and S. N. Nayak, "A Secure Image Encryption and Embedding Approach using MRSA and RC6 with DCT Transformation," *Int. J. Comput. Networks Appl.*, vol. 9, no. 3, pp. 262–278, 2022, doi: 10.22247/ijcna/2022/212553.
[23] E. Liu, A. K. Jain, and J. Tian, "A coarse to fine minutiae-based latent palmprint matching," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 10, pp. 2307–2322, 2013, doi: 10.1109/TPAMI.2013.39.
[24] A. Herbadji, N. Guermat, L. Ziet, Z. Akhtar, M. Cheniti, and D. Herbadji, "Contactless multi-biometric system using fingerprint and palmprint selfies," *Trait. du Signal*, vol. 37, no. 6, pp. 889–897, 2020, doi: 10.18280/TS.370602.

Authors' Profiles

Pradeep R. is currently pursuing his Ph.D. in Computer Science and Engineering from Visvesvaraya Technological University, Belagavi, India. He earned his M.Tech. in Digital Communication and Networking and B.E. in Information Science and Engineering from Visvesvaraya Technological University (VTU), Belagavi, India in the years 2014 and 2012 respectively. He has published papers in various reputed International Journals and Conferences. His research interests include Formal Verification of Security Protocols, Biometric Security Systems, Block Chain and Face Recognition Technology.

Dr. N. R. Sunitha received the B.E. degree from Gulbarga University, the M.S. degree from the Birla Institute of Technology and Science, and the Ph.D. degree from Visveswaraiah Technological University, Belguam. She is currently a Professor with the Department of CS&E, Siddaganga Institute of Technology, India. She has published more than 65 peer-reviewed research articles in leading conferences and journals, such as ACM, Springer, the IEEE, and Elsevier. Her research interests include cryptography and network security, storage area networks, big data processing, industrial automation, and computer security and reliability. She was funded for her research projects from ABB GISL, AICTE, DRDO, IISc, and ICT Skill Development Society in India. She has acquired totally six patents in her research field. Dr. Sunitha possesses membership of personal bodies in the Association of Computing Machinery, USA (ACM), the Indian Society for Technical Education, India (ISTE)—Life Member, the Computer Society of India (CSI), the International Association of Engineers (IAENG), and the Institution of Engineers (FIE). She received the IBM Mentor Award, in 2014. She was a chairperson in the international conferences, such as Conference on Information Science and Technology Management (CISTM 2007), Conference on Network Security and Applications (CSNA 2010), National Conference on Advances in Computer Applications (NCACA), and International Conference on Advances in Computing (ICAdC 2012). Her biodata included in Marquis Who's Who in Science & Engineering 2010. She is a Reviewer of the journals, such as Elsevier's Computers and Security and International Journal of Network Security (IJNS).

Dr. G. S. Thejas is an Assistant Professor at the Department of Computer Science and Electrical Engineering at Tarleton State University (TSU), The Texas A&M University System, since 2020. His research areas include Machine Learning, Deep Learning, Click Fraud Detection, Cybersecurity, Human-Computer Interaction (HCI), and Performance Optimization using Parallel Computing. He directs the Machine Intelligence and Security Research Laboratory (MISR Lab). He worked as a trainee for one year at the Defence Research and Development Organization/Electronic and RADAR Development Establishment (DRDO/LRDE). He worked as an Assistant Professor for five years at Siddaganga Institute of Technology (SIT), India. He is a recipient of the Best Graduate Student in Service Award, Dissertation Year Fellowship Award, two times FIU School of Computing and Information Sciences travel Award, FIU Graduate and Professional Student Committee (GPSC) travel grant, and Graduate Assistantship award at FIU. Thejas's research has successfully produced several papers in top conferences and journals like ACM, IEEE, Springer, Elsevier, and MDPI. He is a member of the Institute of Electrical and Electronics Engineers (IEEE) and the Association for Computing Machinery (ACM). Elsevier News Board: Based on the work entitled "A hybrid and effective learning approach for Click Fraud detection" has been recognized and appeared in Elsevier's news board as "Curbing the clicking con: The automated detection of click fraud", May 25, 2021.

How to cite this paper: Pradeep R., N. R. Sunitha, G. S. Thejas, "A Modern Mechanism for Formal Analysis of Biometric Authentication Security Protocol", International Journal of Computer Network and Information Security(IJCNIS), Vol.15, No.3, pp.15-29, 2023. DOI:10.5815/ijcnis.2023.03.02