electronics

MDPI

Article

MFF-IoT: A Multi-Granularity Formal Framework of User Authentication for IoT

Yuan Fei [1], Jiaqi Yin [2,*] and Lijun Yan [1,*]

[1] College of Information, Mechanical and Electrical Engineering, Shanghai Normal University, Shanghai 201418, China; yuanfei@shnu.edu.cn
[2] School of Software, Northwestern Polytechnical University, Xi'an 710129, China
* Correspondence: jqyin@nwpu.edu.cn (J.Y.); flying@shnu.edu.cn (L.Y.)

Abstract: The Internet of Things (IoT) generates vast amounts of data from numerous applications. However, since wireless channels are the primary means of communication, IoT networks are vulnerable to several security threats, which can compromise their security and privacy. To address these issues, various user authentication protocols have been proposed. Thus, it is still a challenge to provide multi-granularity verifications for different authentications of the IoT. In this paper, we propose a multi-granularity formal framework of user authentication for the IoT (MFF-IoT). Our framework builds different formal models (specification language HLPSL models, process algebra CSP models, Timed CSP models, and timed automata) to complete multi-granularity formal verification. By using both coarse-grained and fine-grained modeling, we can balance the tradeoff between model complexity and verification accuracy. Specifically, our fine-grained models provide a more detailed representation of the framework's behavior and enable us to perform timing-related probability analysis. As these formal models can be implemented by model-checking tools (AVISPA, PAT with C#, and UPPAAL), important properties and features can be analyzed and verified. We also propose several algorithms for better formal model building and evaluate our framework with a case study to show its practicality and effectiveness.

Keywords: formal verification; IoT; user authentication; AVISPA; CSP; PAT with C#; UPPAAL

Citation: Fei, Y.; Yin, J.; Yan, L. MFF-IoT: A Multi-Granularity Formal Framework of User Authentication for IoT. *Electronics* **2023**, *12*, 2356. https://doi.org/10.3390/electronics12112356

Academic Editor: Myung-Sup Kim

Received: 29 March 2023
Revised: 17 May 2023
Accepted: 20 May 2023
Published: 23 May 2023

1. Introduction

The Internet of Things (IoT) is composed of a large number of devices that can connect and exchange data with each other over the Internet or other communication networks. IoT devices are growing at an unprecedented rate, and they are involved in many fields, including homes, offices, transportation, healthcare, etc. Although this situation provides tremendous convenience to our lives, it provides insufficient security due to the immaturity of the technology. As more devices are connected to the Internet, adversaries have more opportunities to use them to launch large-scale attacks [1,2]. Therefore, securing IoT devices is an increasingly serious challenge for manufacturers and consumers. Authentication frameworks and mechanisms [3,4] have been proposed in response to this challenge.

User authentication is an important part of the authentication framework. If there are security holes in user authentication, the risk of being invaded is very serious. Attackers can exploit these holes to carry out malicious activities, such as stealing sensitive data, tampering with device settings, etc. Therefore, verification is required to ensure that the implementation of the authentication framework for IoT satisfies the requirement (or specification) of security. Common methods for the verification process include simulating, testing, or formal verification (e.g., model checking). Formal verification is a method that can describe the system and its requirements formally. It mathematically proves whether the system meets these requirements. Model checking is one of the most popular formal verification techniques, which can automatically verify finite-state machines. Therefore, formal verification is recommended for user authentication in IoT applications.

As model checking helps to avoid manual testing and save time, it is suitable for modeling the authentication framework [5]. However, model-checking-based methods are still not widely used in the IoT. One reason is that it is very difficult for users to build a multi-granularity support model. Although some researchers have conducted formal verification of their schemes [6,7], there is a lack of methods to evaluate performance in a multi-grained way. To solve this problem, a formal verification method should generate multi-granularity formal models from the authentication framework as automatically as possible.

1.1. Contribution

As discussed above, our motivation is to provide a multi-granularity formal framework of user authentication for the IoT (MFF-IoT) that consists of one coarse-grained formal verification and two fine-grained formal verifications. Formal verification is a methodology that employs mathematical techniques to ensure that a system satisfies a predetermined set of properties. Model checking is one such technique, which involves verifying a formal model of the system against a set of formal specifications or properties. The formal verification technique we used is model checking, specifically using the model checkers AVISPA [8], PAT [9] and UPPAAL-SMC [10]. In model checking, the formal model is typically expressed in a formal language such as CSP or Timed CSP, and the model checker systematically explores all possible states of the model to verify whether the specifications hold for all possible system executions. Granularity refers to the level of detail at which the formal model is constructed. In coarse-grained formal verification, some aspects of the authentication framework's behavior are abstracted away, while in fine-grained formal verification, all aspects of the authentication framework are modeled explicitly. We introduce two kinds of fine-grained models; the second fine-grained formal verification places a greater emphasis on timing-related probability analysis than the first fine-grained formal verification.

Figure 1 (Page 3) presents the overall architecture of MFF-IoT, and Figure 2 (Page 3) illustrates a flow chart of MFF-IoT. In the coarse-grained formal verification, we constructed the authentication framework as HLPSL [11] models, which are fed to the AVISPA tool to complete basic security verification. In the first fine-grained formal verification, we use formal language to describe the authentication framework, resulting in corresponding CSP models and Timed CSP models, which are then implemented using the model checker PAT with C#. The CSP models and Timed CSP models are checked against linear temporal logic (LTL) properties. In the second fine-grained formal verification, we construct timed automata in the UPPAAL-SMC tool for several components in the authentication framework to verify time-related security probabilistic properties, thus obtaining observable statistics in several different probability settings.

Moreover, we use the gateway-based two-factor authentication (G2F) framework [12] as a case study to show the practicality and effectiveness of MFF-IoT. We propose an algorithm for coarse-grained formal verification to construct HLPSL models for the entities in the G2F framework. We also design two algorithms for the entity's CSP model building and Timed CSP model building in the first fine-grained formal verification. For the second fine-grained formal verification, our idea is to model key actions using timed automata in the G2F framework from the perspectives of the involved entity (user or intruder).

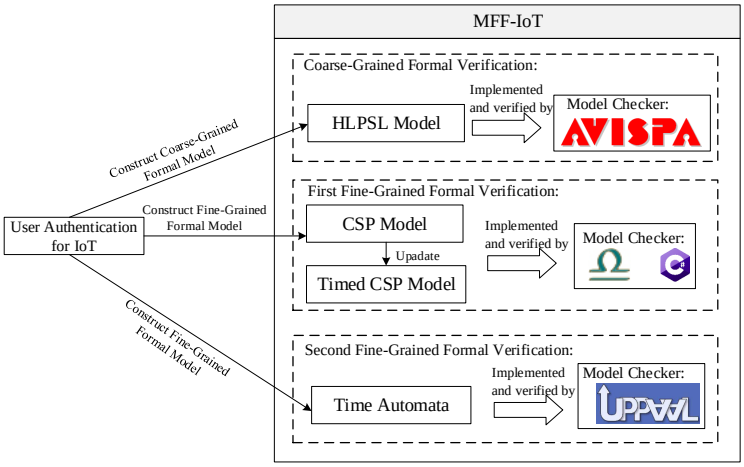


Figure 1. The overall architecture of MFF-IoT.

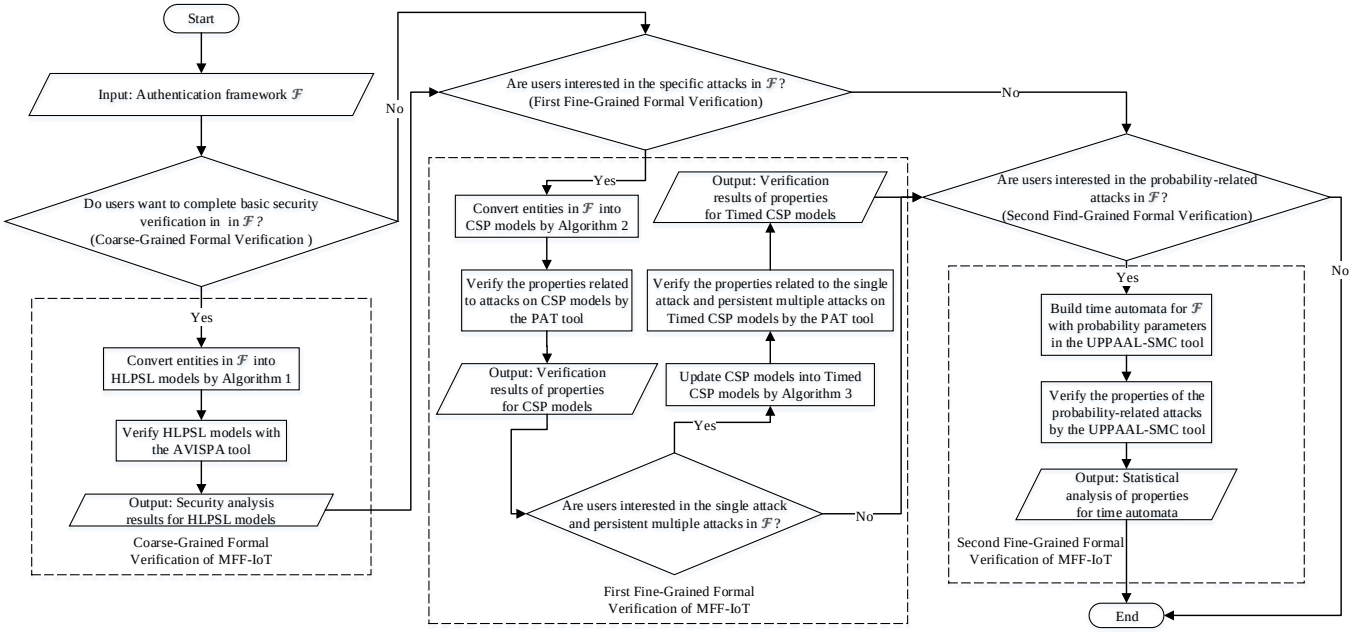


Figure 2. Flow chart of MFF-IoT.

1.2. Related Work

As a promising method for security assurance, formal verification can apply different kinds of mathematical and logical methods to verify the correctness of designs. It is used in many systems that require safety and security properties. Formal verification also has a variety of applications in the IoT, such as authentication and communication protocols.

Shkarupylo et al. [5] proposed an approach to check the interoperability between the components of IoT systems. Using Temporal Logic of Actions (TLA), TLA+ and PlusCal formalisms, as well as the corresponding TLC (model checker for TLA), the MQTT application IoT protocol is analyzed as a case study. Hofer-Schmitz and Stojanovic [13] reviewed formal methods for a large variety of protocols used in the IoT environment, such as ZigBee, Z-Wave, 6LoWPAN, Sigfox, and Narrowband-IoT. Aziz [14] modeled and analyzed the MQ Telemetry Transport version 3.1 protocol based on a timed message-passing process algebra. They found some potential vulnerabilities and suggested an enhancement. Mohsin et al. [15] introduced a formal framework called IoTSAT using SMT logic. The framework can create a satisfiable (SAT) answer, which carries the threat

resilience and presents the potential threat vectors. Mahadewa et al. [16,17] presented an approach that checks the security of the implementations of IoT systems by extracting the abstract specification of the application-layer protocol and internal behaviors. They discovered twelve security vulnerabilities in three IoT systems. Aktas and Astekin [4] proposed a run-time formal verification mechanism to support self-healing IoT applications that can detect faulty running behavior in IoT devices. These studies provided a variety of verifications for specific IoT authentication but failed to provide a general verification supporting multi-granularity. Our work introduces the verification of various security properties in a multi-grained way.

The rest of this paper is organized as follows. Section 2 presents the related background. Section 3 provides the architecture of our methodology. Section 4 applies MFF-IoT in a case study. Finally, Section 5 concludes the paper and discusses future work.

2. Background

In this section, we present process algebra CSP and Timed CSP. Model-checking tools AVISPA, PAT, and UPPAAL are also introduced. A case study for MFF-IoT is also presented.

2.1. CSP and Timed CSP

As one of the most mature formal methods, communicating sequential processes (CSP) [18,19] is tailored to describe the interaction between concurrency systems via mathematical theories. Because of its well-known expressive ability, CSP has been widely used in many fields [20,21]. CSP processes are constituted by primitive processes and actions. We use the following syntax to define the processes in this paper, where P and Q represent processes, and $\alpha(P)$ and $\alpha(Q)$ indicate the sets of actions that processes P and Q can take, respectively. Hence, a and b denote atomic actions, and c represents a channel.

$$P,Q ::= ... \mid SKIP \mid a \rightarrow P \mid c?x \rightarrow P \mid c!e \rightarrow P \mid P \| Q \mid P;Q$$

where *SKIP* is process that only terminates successfully. $a \rightarrow P$ first performs action a, then behaves like P. $c?x \rightarrow P$ receives a message from channel c and assigns it to variable x, then implements the subsequent behavior like P. $c!e \rightarrow P$ sends a message (e) through channel c, then performs P. $P \parallel Q$ shows the parallel composition between P and Q. P;Q represent the execution of P and Q, respectively.

Timed CSP [22] is an extension of CSP with a time concept. It is defined as follows.

$$P,Q ::= ... \mid WAIT\ t \mid P \rhd^t Q$$

where *WAIT t* is a delayed form of *Skip* and only terminates after the specified time (t). The notation $P \rhd^t Q$ indicates that if process P is not executed within a certain time interval of t time units, process Q will be executed at time t. In other words, $P \rhd^t Q$ specifies a constraint on the maximum duration of the execution of process P, and if this constraint is violated, process Q is activated to perform a recovery action or to initiate a new activity. It is useful in modeling real-time systems in which processes need to be completed within specific time constraints.

2.2. AVISPA, PAT, and UPPAAL

The Automated Validation of Internet Security Protocols and Applications (AVISPA) tool [8] provides a modular and expressive formal language for specifying protocols and security properties. It integrates different back ends that implement a variety of automatic protocol analysis techniques. It can also automatically translate the High-Level Protocol Specification Language (HLPSL) [11] specification into an equivalent specification written in the rewrite-based formalism (Intermediate Format IF), then obtain the checking results. AVISPA has been successfully employed in some classic and familiar protocols and schemes [8,23].

As a model-checking tool, the Process Analysis Toolkit (PAT) [9] was designed as an extensible and modularized framework based on CSP, which can also support Timed CSP. Different model-checking techniques are implemented in PAT, supporting many assertions, such as deadlock freeness and reachability [24]. With advanced optimization techniques implemented in PAT, it can achieve satisfactory performance.

UPPAAL is an integrated tool for the modeling, simulation, and verification of real-time systems [10]. It is appropriate for verifying systems that can be modeled as a collection of non-deterministic processes with finite control structures and real-valued clocks, communicating through channels or shared variables. UPPAAL-SMC is an extension of UPPAAL that can reason on networks of complex real-time systems under natural stochastic semantics, which is based on the idea of statistical model checking (SMC). UPPAAL has achieved fruitful results in industry [25] and achieves satisfactory performance in protocol verification [26]. Therefore, these three powerful tools can effectively support our multi-granularity formal verification method.

2.3. Case Study: Gateway-Based Two-Factor Authentication (G2F)

Lou et al. [12] introduced a secure user authentication for IoT management called G2F. The whole authentication scheme is based on the Universal Second Factor (U2F) protocol proposed by the FIDO Alliance. G2F can prevent malicious activity on IoT devices hosted on cloud servers, which also helps protect IoT devices from malicious attacks. It is an efficient, secure, and flexible user authentication solution for IoT-based smart homes, which makes it a suitable case study. Our proposed framework can also be applied to other user authentication schemes, such as those used in IoT-based healthcare [27]. The G2F system consists of five components:

- IoT Server: The IoT Server offers various IoT services, such as remote control and configuration of IoT devices;
- U2F Server: The U2F server works in coordination with the hardware token to facilitate registration and authentication processes. It is responsible for storing crucial security elements related to the hardware token, including public keys, key handles associated with public keys, etc;
- Gateway Node (GWN): The GWN serves as a bridge connecting IoT devices to a cloud server. Its main function is to gather data from IoT devices and transmit them to the cloud server for processing. Additionally, the GWN is equipped with a USB interface, which allows for the insertion of a hardware token;
- Hardware Token: The hardware token is a highly secure component held by the user and designed to be tamper-proof. It contains private keys and other security elements and communicates with the GWN through a USB interface. When registration or authentication requests are made, users can respond by pressing the button on the token;
- IoT Device: IoT devices are owned by the user and are managed through the IoT Server. They connect to the GWN wirelessly to access network services. The client GUI of the IoT server allows users to perform operations on their IoT devices.

The G2F system is comprised of three levels: the cloud server, the gateway node, and the user. The U2F server and IoT server belong to the cloud server level, while the hardware token and GWN belong to the gateway node level. The IoT device and the client belong to the user level.

The G2F framework includes the registration phase and authentication phase. We only introduce the authentication phase because most attacks occur in this phase, and our work mainly analyzes this part. User authentication is needed when the user wants to start operations on the IoT device through the IoT server. A detailed description of the G2F

authentication phase is presented in Table 1, with explanations of the notations that appear in the authentication phase.:

Step 1. IoT Server → GWN : notify GWN to start authentication;
Step 2. GWN → U2F Server : submit authentication request;
Step 3. U2F Server → GWN : {*handle*, *app_id*, *challenge*};
Step 4. GWN : verify *app_id*;
Step 5. GWN → Hardware Token : {*handle*, *app_id*, *challenge*, *origin*, *channel_id*};
Step 6. Hardware Token : find K_{priv} linked with *h*; *counter*=*counter* + 1;
Step 7. Hardware Token → GWN :
{*counter*, [*app_id*, *challenge*, *origin*, *channel_id*, *counter*]K_{priv}};
Step 8. GWN → U2F Server : {*counter*, *challenge*, *origin*, *channel_id*, *signature*};
Step 9. U2F Server : verify *signature* using K_{pub} linked with *handle*;
verify *origin*, *channel_id*, *counter*;
Step 10. U2F Server → IoT Server : feed back the result;
Step 11. IoT Server : check whether authentication is passed;
Step 12. IoT Server → IoT Device : operation on the IoT device.

Table 1. Explanations of notations in the authentication phase of the G2F framework.

Notation	Explanation
app_id	The identity of the IoT application
challenge	A random number created by the U2F server
channel_id	The identity of the transport-layer security (TLS) channel
origin	The uniform resource identifier (URI) of the U2F server
k_{pri}, k_{pub}	The private key and public key generated by the hardware token
counter	The counter counting authentication times
handle	The key handle created by the hardware token
signature	The signature itself or the signing operation produced by the hardware token

If the IoT server is informed to start an operation, it tells the GWN to send an authentication request to the U2F server; then, GWN sends an authentication request to the U2F server (steps 1–2). The U2F server replays {*handle*, *app_id*, *challenge*} to the GWN; then, the GWN checks *app_id* and sends {*handle*, *app_id*, *challenge*, *origin*, *channel_id*} to the hardware token (steps 3–5). The counter is increased when a *signature* is generated (i.e., the hardware token signs {*app_id*, *challenge*, *origin*, *channel_id*, *counter*} by k_{pri}), and k_{pri} can be obtained based on *handle* (step 6). Then, the hardware token transmits {*counter*, *signature*} to the GWN (step 7). The GWN sends this message, together with the message it previously received from the hardware token, to the U2F server (step 8). The U2F server verifies the signature using the public key k_{pub} (step 9). The verification result is returned to the IoT server (step 10). If the user authentication is successful, the IoT server allows the operation on the IoT device (steps 11–12).

2.4. Threat Model

As all communication channels are unencrypted, attackers can obtain all messages. While they cannot obtain the hardware token, they can conduct a hardware token faking attack using their own public and private keys to forge signatures. Attackers can perform both single and persistent multiple attacks. In a single attack, they execute the hardware token faking attack only once within a designated time frame, assuming there are no normal

user operations that need to be executed during this period. In a persistent multiple attack, they continuously execute hardware token faking attacks within a designated time frame, assuming there are normal user operations that need to be executed during this period.

3. The Architecture of Our Methodology

In this section, we introduce MFF-IoT, as illustrated in Figure 1, including coarse-grained formal verification, a first fine-grained formal verification, and a second fine-grained formal verification. Figure 2 shows a flow chart of MFF-IoT.

3.1. Representation of the Authentication Framework

In the process of modeling, whether at a coarse-grained or fine-grained level, it is necessary to model the entities of the authentication framework ($\mathcal{F}$) and specifically model the actions performed by these entities. In $\mathcal{F}$, most actions involve communication, while a small portion involves processing. The former often involves sending information, including signatures, to another entity, while the latter involves processing counters or performing signature-related behaviors. As an example of communication action, in coarse-grained modeling, keywords such as RCV and SND from communication protocols can be used to describe the corresponding receiving and sending of transmitted content. In fine-grained modeling, channels can be used for message transmission and reception.

3.2. Coarse-Grained Formal Verification in AVISPA

In the coarse-grained formal verification, we propose the use of the HLPSL algorithm to model the entity role in any authentication framework using the AVISPA tool. The overall steps can be roughly summarized as follows:

1. Divide the roles according to the entities in the authentication framework;
2. Define local variables;
3. Set different transition states;
4. For transmission actions, the actions in the transition are defined as assignment, sending, and receiving.

Further details can be found in Algorithm 1. After applying Algorithm 1 to generate the HLPSL model of each entity in the authentication framework, we adopt the homologous OFMC tool in AVISPA to verify the authentication property between entities and provide a general but coarse-grained conclusion as to whether a basic intrusion is detected, which indicates which element is most likely to cause intrusion.

3.3. First Fine-Grained Formal Verification in CSP and Timed CSP

In the coarse-grained formal verification, we only focus on the knowledge domain of the intruder and pay no attention to attack types. To solve this problem, we propose the first fine-grained formal verification. Algorithm 2 can be applied to build the CSP model of each entity. By combining different CSP models of entities with the parallel composition operator, the overall model for the authentication framework can be described.

Then the CSP models can be implemented in PAT and C#. Because PAT is extended based on CSP, the channel output/input in PAT can also be used to implement message sending and receiving in CSP. Meanwhile, functions should be implemented to complete the functional actions in the CSP model. As it is difficult to write complex functions in PAT, PAT allows for the use of C# code for libraries. C# classes are built as DLL and imported into the PAT model when the functions are used. As a result, we choose to implement our functions in the C# language. To convert the CSP model to the PAT model, the PAT model calls the C# functions, which have implemented processing actions, completing the integration of C# into PAT to support modeling and verification.

After the CSP models are implemented in PAT and C#, we can verify some properties, such as the deadlock property and the property indicating whether digital signatures can be verified under attacks. These properties can be described as linear temporal logic (LTL) formulations and verified using PAT. To support the verification of single attacks and

persistent multiple attacks, we update the overall CSP model in the overall Timed CSP model. As a result, the CSP model of the entity involving time can be updated to the Timed CSP model using Algorithm 3. A renaming operation is used to support the description of the intruder. The *WAIT* operator in timed CSP can simulate the waiting time. With the help of this operator, the overall Timed CSP model can be constructed. The subsequent verification is still conducted in PAT in the same way, which is also able to detect single attacks and persistent multiple attacks.

Algorithm 1 Entity's HLPSL Model Building Algorithm

Input: entity name e, entity's key set K,
entity's local known elements set E, entity's steps list S.
the sequence number of the steps list of this entity N
Output: string of entity's HLPSL model p.

Extract messages in S into message set M and communication entity set M'
$p \leftarrow$ "*role role_*"+e+"("+e+"*:agent*"+
elements in E separated with "*:text*,"
if $K \notin \emptyset$ **then**
$p \leftarrow p$+elements in K separated with
"*:public_key*,"+"*SND*, *RCV*:*channel*(*dy*))"
else
$p \leftarrow p$+"*SND*, *RCV*:*channel*(*dy*))"
end if
$p \leftarrow p$+"*played_by HardwareToken*"
$p \leftarrow p$+"*def=local State:nat*,"+
elements in M separated with "*:text*,"+
elements in S separated with "*:agent*"
Choose the relevant sequence number of the steps to assign to the transition state
$p \leftarrow p$+"*init State*:="+choose the relevant number N_i in N
Find the elements changed in M to form M'
$p \leftarrow p$+"*transition*"+choose the relevant number N_j in N + ". *State*=$N_i \wedge$ *RCV*("+elements in S with " ' "+e+"."+messages in M-M' separated with " ' "+") = | >"+"*State'*:="+choose the relevant number N_k in N + elements in M' with " ' "+":=*new*()"
if we need to verify the authentication property between e and the entity in S **then**
Define the authentication property *auth*
$p \leftarrow p$+"$\wedge$ *witness*("+e+","+elements in S +
"," + *auth* + "," + elements in M' with " ' "+")"+
"$\wedge$ *SND*("+e+"."+elements in S with " ' "+"."+
elements in E+"."+elements in M' with " ' "+
"*_inv*("+elements in K+")) *end role*"
else if we request to check the authentication property between e and the entity in S **then**
Define the authentication property *auth*
$p \leftarrow p$+"$\wedge$ *request*("+e+","+elements in S +
"," + *auth* + "," + elements in M' with " ' "+")"+
"$\wedge$ *SND*("+e+"."+elements in S with " ' "+"."+
elements in E+"."+elements in M' with " ' "+
"*_inv*("+elements in K+")) *end role*"
else
$p \leftarrow p$+"$\wedge$ *SND*("+e+"."+elements in S with " ' "+
"."+elements in E+"."+elements in M' with
" ' "+"*_inv*("+elements in K+")) *end role*"
end if
return p

Algorithm 2 Entity's CSP Model Building Algorithm

Input: entity name e, channel names set C,
entity's local known elements set E, entity's steps list S.
Output: string of entity's CSP model p.

Extract messages in S into message set M
$p \leftarrow e$+"("+
elements in E separated with commas+") $=_{df}$"
while $s \in S$ **do**
  if s is a transferring action between e and entity e' **then**
    Find channel c in C connecting e and e'
    Find message m in M corresponding to s
  end if
  if s is a sending action **then**
    Add $'$ suffix to elements in m not belong to E
    $p \leftarrow p$+c+"?"+m+"$\rightarrow$"
  else if s is a receiving action **then**
    Add " $'$ " suffix to all the elements in m
    string $p \leftarrow p$+c+"!"+m+"$\rightarrow$"
  else
    Create n as processing action's name of s.
    $p \leftarrow p$+n+"$\rightarrow$"
  end if
end while
$p \leftarrow p$+"*SKIP*"
return p

Algorithm 3 Entity's Timed CSP Model Building Algorithm

Input: string of entity's CSP model p, times t_1 and t_2.
Output: string of entity's Timed CSP model tp.

Divide p into process head ph and process body pb by searching for a pattern which is "$=_{df}$".
if there are operations in pb that need to be executed within time t_1 **then**
  Divide pb into two disjoint parts pb_1 and pb_2, where pb_1 has no time limit for execution and pb_2 has a time limit for execution.
  $pb \leftarrow pb_1$+"$\rightarrow$"+ "("pb_2+$\triangleright^{t_1}$+"*SKIP*)"
end if
if pb needs to wait for time t_2 before executing **then**
  $pb \leftarrow$"*WAIT* t_2; ("+pb+")"
end if
$tp \leftarrow ph$+" $=_{df}$"+pb
return tp

3.4. Second Fine-Grained Formal Verification in UPPAAL-SMC

The first fine-grained formal verification is used to analyze important security properties, which can help to detect security vulnerabilities. However, different time parameters may lead to different results, and the first verification may not be able to further analyze more details. If further probability analysis is needed, we propose a second fine-grained formal verification, which is supported by UPPAAL-SMC. The second verification focuses on situations in which an entity may receive messages from both an intruder and the other entity and involves analysis of the complex interactions between different components to detect security vulnerabilities that may arise due to these interactions.

Specifically, we focus on a situation in which an entity may receive messages from both the intruder and the other entity. According to the action being performed by the entity in the situation, multiple automata are built. They generally simulate the key behaviors of

the current entity, the user, and the intruder. The characterization of probability and time is added to the automaton so that UPPAAL-SMC can be applied to verify the properties of the probability-related attacks, which occur in the form of cost-constraint temporal logic formulations.

We chose different verification schemes based on the specific aspects that we want to verify. For example, the coarse-grained formal verification scheme only focuses on the knowledge domain of the intruder. However, for more detailed aspects such as counter changes and timing, the first fine-grained formal verification scheme is necessary. Additionally, if we want to analyze the probabilities of time-related events, the second fine-grained formal verification scheme becomes indispensable. The AVISPA tool is not suitable for capturing the details of element changes, so we use CSP and timed CSP, which provide support for time and element changes, and UPPAAL-SMC, which supports the analysis of time-related probabilities.

4. Application of MFF-IoT in the G2F Framework

In this section, we apply MFF-IoT in the G2F framework. By modeling the G2F framework in HLPSL models, CSP models, Timed CSP models, and timed automata, security properties are described and verified. We also analyze and discuss the verification results.

4.1. Coarse-Grained Formal Verification for G2F

We provide the coarse-grained formal verification for the G2F framework. The AVISPA tool is used to model the authentication phase of the G2F framework. The security of this phase is analyzed below.

4.1.1. Modeling the G2F Framework in AVISPA

The authentication phase of G2F involves five entities: the hardware token, GWN, U2F server, IoT server, and IoT service. Henceforth, we model this phase based on these five entities' roles. In addition, an entity role intruder is added to our HLPSL model, which knows all the messages except the hardware token. Here, we also only provide the HLPSL model of the hardware token in AVISPA as an example, which is shown in Table 2. In this phase, the hardware token can communicate with the entity role *GWN* so that the input in Algorithm 1 can have the entity name *HardwareToken*. Similarly, we can provide other content of the input. Hence, the model of the entity role *HardwareToken* shown in Table 2 illustrates that it has one state to be changed. We also add an intruder into the model and test all the kinds of the knowledge domain. Finally, we know that if we can protect the security and privacy of the hardware token, the safety of the phase can also ensured, no matter what other messages the intruder knows. Please refer to the website (The full implementation in AVISPA can be found in https://github.com/asunafy/MFF/tree/main/AVISPA (accessed on 28 March 2023)) for full implementation in AVISPA.

Table 2. The HLPSL model of the hardware token.

role role_HardwareToken(*HardwareToken* : *agent*, *Counter* : *text*, *K* : *public_key*, *SND*, *RCV* : *channel*(*dy*))	(1)
played_by HardwareToken	(2)
def =	(3)
local	(4)
State : *nat*, *ChanID* : *text*, *Challenge* : *text*, *Handle* : *text*, *AppID* : *text*, *Origin* : *text*, *SSS* : *text*, *GWN* : *agent*	(5)
init	(6)
State := 0	(7)
transition	(8)
4. *State* = 0 ∧ *RCV*(*GWN'*.*HardwareToken.Handle'*.*AppID'*.*Challenge'*.*Origin'*.*ChanID'*) = | >	(9)
State' := 1 ∧ *SSS'* := *new*() ∧ *witness*(*HardwareToken*, *GWN*, *auth_1*, *SSS'*)∧	(10)
SND(*HardwareToken.GWN'*.*Counter*.{*SSS'*}_*inv*(*K*))	(11)
end role	(12)

4.1.2. Security Verification

We adopt the AVISPA tool to verify the security of the authentication phase, and its analysis result is presented in Figure 3. We enable OFMC (on-the-fly model checker) in AVISPA to verify the identity authentication between the GWN and the hardware token as the goal in order to verify the security of the G2F framework design. The statistics presented in Figure 3 show that the state search time during the entire verification process is 1.23 s, with 648 visited nodes and a depth of 11 plies, and the summary is safe.

In particular, we assume that both intruders and normal entities can ensure the normal operation of the function. According to the analysis result, we believe that as long as the hardware token is not leaked, even if the intruder knows all other information and has public and private keys to intervene, the security of this protocol can still be guaranteed. That is to say that the hardware token is held by the user, so we can infer that the design of the G2F framework for the authentication phase is secure. Thus, we can conclude that the G2F framework is capable of resisting hardware token faking attacks, assuming that the signature content is not taken into consideration and the hardware token has not been lost.

```
% OFMC                                  BACKEND
% Version of 2006/02/13                    OFMC
SUMMARY                                 COMMENTS
  SAFE                                  STATISTICS
DETAILS                                   parseTime: 0.00s
  BOUNDED_NUMBER_OF_SESSIONS              searchTime: 1.23s
PROTOCOL                                  visitedNodes: 648 nodes
  /home/AVISPA/Authentication.hlpsl       depth: 11 plies
GOAL
  as_specified
```

Figure 3. Analysis result of authentication phase in AVISPA.

4.2. First Fine-Grained Formal Verification for G2F

In the coarse-grained formal verification, we only focus on the knowledge domain of the intruder and pay no attention to attack types and signatures. Our first fine-grained formal verification supports the discussion provided in this section. Specifically, we provide the CSP models of entities to build the system and update them into the Timed CSP model to discuss the performance of the G2F framework under a single attack and persistent multiple attacks. They are also implemented in PAT and C#. Finally, the model-checking tool PAT is applied to verify security properties (deadlock freedom, signature verification, and intruder success).

4.2.1. Modeling the G2F Framework in CSP

To simplify the model, we focus on the steps involving the transmission of key information in the authentication phase of the G2F framework, which are shown in Steps 3–9. There are three entities involved in this process: the hardware token, GWN, and U2F server. The related CSP models are built by Algorithm 2. Then, we can obtain the overall model (*G2F_Auth*()) with the parallel composition operator. We then study a hardware token faking attack in which the hardware token is replaced by a fake token. Its CSP model is *G2F_Auth_S*(), which is almost the same as *G2F_Auth*(), except that the parameter of the signature is replaced by a fake parameter.

We explain the CSP model of the hardware token (*HT_Auth*()) as an example, which is illustrated in Table 3. This process describes the behaviors of the hardware token when communicating with the GWN in the authentication phase. It first obtains *handle*, *app_id*, *challenge*, *origin*, and *channel_id* from the GWN (step6). Through the action *Increase_Counter* and *Generate_Signature*, it increases the *counter* and creates *signature* using the private key included in *handle* (step7). Finally, it transmits *counter* and *signature* to the GWN (step8).

Table 3. The CSP model and Timed CSP model of the hardware token.

$HT\_Auth(counter, signature) =_{df}$	(1)
$ComGH\_U?msg_{auth}.handle'.app\_id'.challenge'.origin'.channel\_id' \rightarrow$	(2)
$Increase\_Counter \rightarrow Generate\_Signature \rightarrow$	(3)
$ComGH\_U!msg_{auth}.counter.signature \rightarrow SKIP$	(4)
$HT\_Auth'(counter, signature) =_{df}$	(5)
$WAIT\ t; (ComGH\_U?msg_{auth}.handle'.app\_id'.challenge'.origin'.channel\_id' \rightarrow$	(6)
$Increase\_Counter \rightarrow Generate\_Signature \rightarrow$	(7)
$ComGH\_U!msg_{auth}.counter.signature \rightarrow SKIP)$	(8)

4.2.2. Updating CSP Models to Timed CSP models

To discuss the performance of the G2F framework under a single attack and persistent multiple attacks [12], we upgrade the CSP model to the Timed CSP model to support appreciably finer-grained formal verification. Table 3 also shows the Timed CSP model of the hardware token (*HT_Auth'*()). We also verify two properties (deadlock freedom and intruder success) under the two attacks for the new models (*G2F_Auth_Single*() and *G2F_Auth_Multiple*()).

In the single attack and persistent multiple attacks, intruders request that IoT servers perform a malicious operation. Whether step 7 and the subsequent steps are performed depends on whether the user presses the button on the hardware token. In a single attack, we assume that the user does not press the button within the time consumption of the authentication phase. For persistent multiple attacks, the user may submit a legitimate request during the period during which the intruder continues to send malicious operation requests. Then, the user presses the button for his legitimate request, which may cause the malicious operation request to be passed. Equipped with Algorithm 3, we complete the conversion from the CSP models involving time to Timed CSP models.

4.2.3. Property Verification

We implemented CSP models and Timed CSP models in the PAT tool (The full implementation of the CSP models in PAT and C# can be found in https://github.com/asunafy/MFF/tree/main/PAT (accessed on 28 March 2023)). The implementation capability of PAT for data processing is limited, and using its C# interface would be beneficial to perform additional actions, such as making signatures. Moreover, C++ provides better support for random number generation, so we can import C++ functions into C# code. Specifically, to implement the steps belonging to processing actions, we introduce one C++ function and five C# functions. The C++ function is built as DLL, then imported into the C# class. The six functions implement random number generation, type conversion, file storage, file reading, signature generation, and signature verification, respectively. We provide three properties to be verified in Table 4.

- Deadlock freedom property (in Table 4(1)): All the CSP and Timed CSP models should have no deadlock. "deadlockfree" is the reserved keyword of PAT. Here, *System*() refers to all models. The valid verification results of the deadlock freedom property presented in Figure 4a and Table 5 indicate that no deadlock appears in our models. In model checking, only models without deadlocks can be further analyzed.
- Signature verification property (in Table 4(2–4)): In the coarse-grained verification, by default, all signatures can be successfully verified by the U2F server, so the signature verification property is not considered. Here, we verify the signature verification property in the CSP model (*G2F_Auth*() and *G2F_Auth_S*()). The property of signature verification is described in the LTL formula, which illustrates a linear-time property. PAT supports the LTL formula by using #assert $P()$ $|=$ F to check whether system $P()$ satisfies the LTL formula F. Using the "eventually" operator ($<>$) in LTL, we describe a situation in which the U2F server checks the signature successfully. As the *G2F_Auth_S*() introduces signature-faking attacks, the signature verification property can effectively detect these attacks. As shown in Figure 4b, the valid verification

result of *G2F_Auth*() indicates that the G2F framework can successfully verify the signature under normal operation, and the invalid verification result of *G2F_Auth_S*() means that our methodology is able to detect illegal signatures to protect the user's legal authentication. Therefore, we can conclude that G2F can effectively identify the signature of a hardware token forged by an intruder.

- Intruder success property (in Table 4(5–7)): The intruder success property checks whether our model can be invaded under single attacks and persistent multiple attacks and is expressed as reachability. Because Timed CSP models *G2F_Auth_Single*() and *G2F_Auth_Multiple*() introduce single attacks and persistent multiple attacks, respectively, with the reserved keywords "*reaches*", PAT can check whether the system reaches a state at which some given condition is satisfied. For *G2F_Auth_Single*(), the four invalid results presented in Table 5 indicate that the G2F framework can prevent a single attack under each of the four cases. For *G2F_Auth_Multiple*(), different verification results appear under different cases in Table 5, indicating that the G2F framework cannot always prevent persistent multiple attacks and that intruders cannot invade successfully every time. Therefore, we can conclude that once introduced into an uncertain environment, there is a possibility that G2F may encounter an intrusion when the timing of an attacker sending intrusion data is close to the timing of user usage.

Table 4. Properties in PAT.

#assert System() deadlockfree;	(1)
#define SignatureCheck (signaturecheck == true);	(2)
#assert G2F_Auth() |=<> SignatureCheck;	(3)
#assert G2F_Auth_S() |=<> SignatureCheck;	(4)
#define IntruderSuccess (intrudersuccess == true);	(5)
#assert G2F_Auth_Single() reaches IntruderSuccess;	(6)
#assert G2F_Auth_Multiple() reaches IntruderSuccess;	(7)

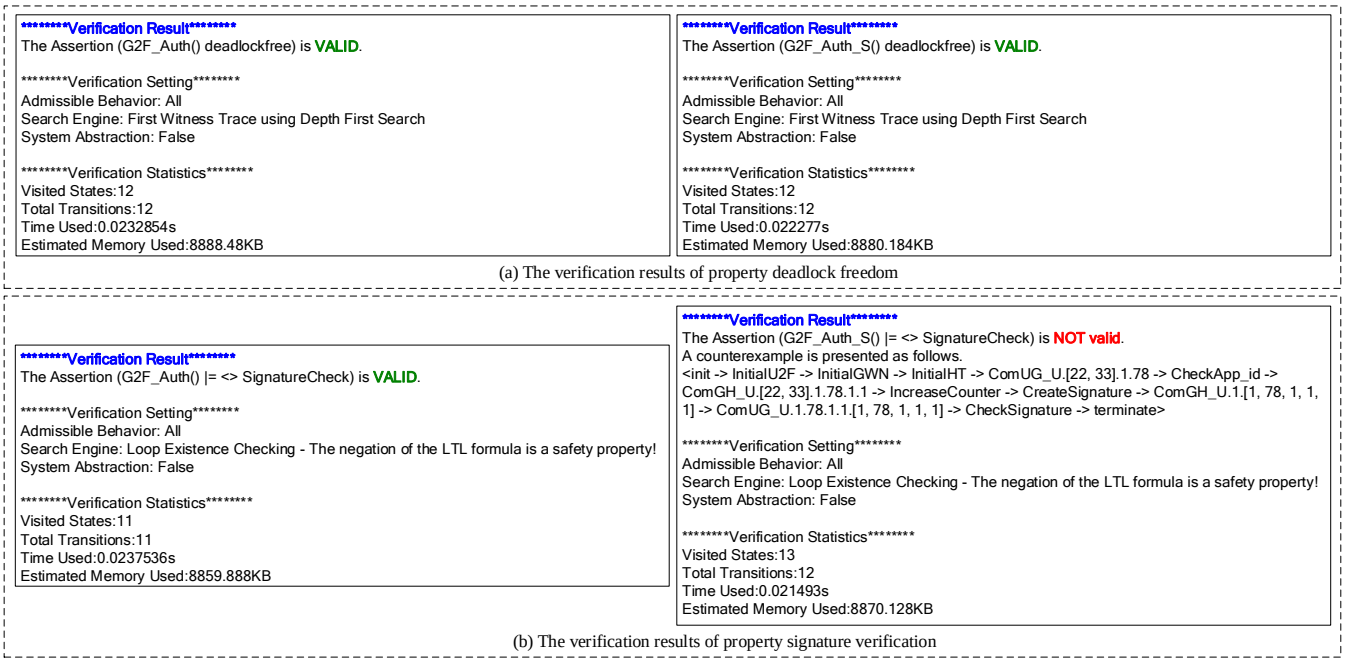


Figure 4. The verification results of properties for CSP models.

Table 5. The verification results of properties for Timed CSP models.

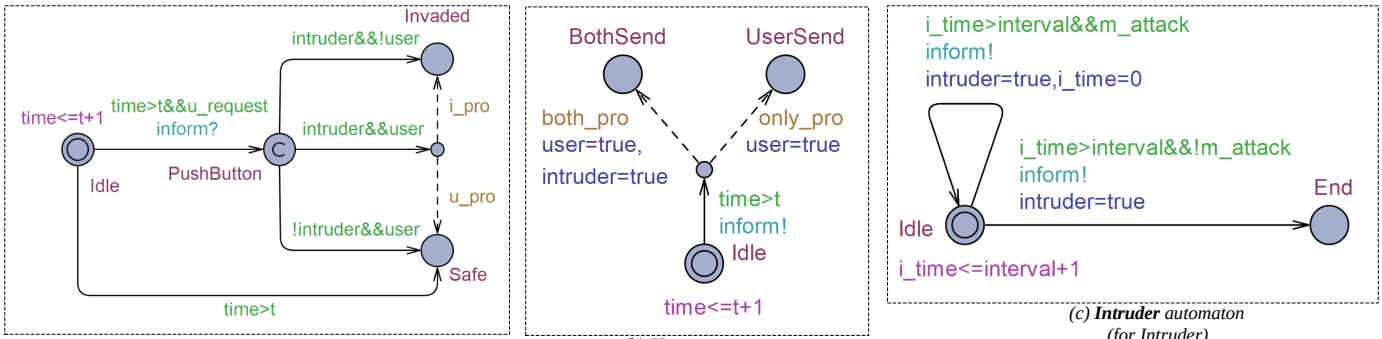
Time Parameter Values				Model Verification Results			
				Deadlock Freedom Property		Intruder Success Property	
TU	TOU	TI	TOI	*G2F_Auth_Single*()	*G2F_Auth_Multiple*()	*G2F_Auth_Single*()	*G2F_Auth_Multiple*()
4	5	3	5	Valid	Valid	Not Valid	Valid
7	5	3	5	Valid	Valid	Not Valid	Not Valid
10	5	3	5	Valid	Valid	Not Valid	Valid
15	5	3	5	Valid	Valid	Not Valid	Not Valid

4.3. Second Fine-Grained Formal Verification for G2F

According to the verification results of Timed CSP models, the intruder may invade the G2F framework when persistent multiple attacks happen, but more detailed information is unknown, such as the probability of being intruded. To further improve the granularity of our formal verification, we use UPPAAL-SMC to model and verify the G2F framework under a single attack and persistent multiple attacks.

4.3.1. Building Timed Automata with Probability Distributions

For persistent multiple attacks, we focus on a situation in which the hardware token receives a notification both from the user and the intruder simultaneously when its button is pushed. Therefore, the behavior of the G2F framework under persistent multiple attacks can be simplified to three automata (The full implementation in UPPAAL-SMC can be found at https://github.com/asunafy/MFF/tree/main/UPPAAL (accessed on 28 March 2023)) (*HT* automaton, *User* automaton, and *Intruder* automaton), as shown in Figure 5. *HT* automaton denotes a situation in which the hardware token receives notifications from the user or intruder, then completes the push-button action. *User* automaton indicates a situation in which the user sends a notification to the hardware token. *Intruder* automaton represents a situation in which the intruder sends notifications to the hardware token periodically. Here, we explain some of the functions implemented in *HT* automaton shown in Figure 5 as an example. Edges among *Idle*, *PushButton*, *Invaded*, and *Safe* with the branch illustrate the user and intruder requests are both received when the user pushes the button. The second Boolean condition (*intruder*&&*User*) means that the hardware token has received the requests from both the user and the intruder. Meanwhile, the *Invaded* branch state is with probability weight *i_pro*, and the *Safe* branch state is with probability weight *u_pro*.



(a) ***HT*** automaton (for Hardware Token)

(b) ***User*** automaton (for User)

(c) ***Intruder*** *automaton (for Intruder)*

Figure 5. *HT*, *User*, and *Intruder* automata.

4.3.2. Probability Property Verification

Timed automata with probability distributions are implemented by UPPAAL-SMC, which can reason on networks of complex real-time systems under natural stochastic semantics. *User_multiple*, *Intruder_multiple*, and *HT_multiple* are instantiations of automata *User*, *Intruder*, and *HT*, respectively, for persistent multiple attacks. At the same time, *Intruder_single* and *HT_single* are instantiations of *Intruder*, and *HT* automata, respectively for a single attack and are included in the system for a single attack. Thus, we can verify the probability of the G2F framework being safe or being invaded.

The four queries in Table 6 are the properties of security probability. They estimate the four probabilities that *HT_multiple* and *HT_single* will reach a *Safe* state or *Invaded* state before 100 time units.

- Probability properties of single attacks: According to the four different probability settings shown in Figure 6, the query in Table 6(1) has the same answer [0.901855, 1] with a confidence of 0.95 in 29 runs. Meanwhile, the query in Table 6(2) has the same answer [0, 0.0981446] with a confidence of 0.95 in 29 runs. The results indicate that single attacks fail in most cases. Therefore, we can conclude that in most cases, a single attack will not pose a threat to the security of the G2F framework.
- Probability properties of persistent multiple attacks: Using the four probability settings in Figure 6, the answers of the query in Table 6(3) are [0.499902, 0.599848], [0.757374,0.857253], [0.393781, 0.49372], and [0.822723, 0.922673], with 95% confidence intervals obtained from 398, 254, 397, and 182 runs, respectively. Meanwhile, the answers to the query in Table 6(4) are [0.421586, 0.521483], [0.151334, 0.251058], [0.473541, 0.573489] and [0.0619394, 0.16173], with 95% confidence intervals obtained from 401, 263, 401, and 163 runs, respectively. These results show that *HT_multiple* has a higher probability of reaching a *Safe* state than reaching an *Invaded* state within the given time limit, except when probability parameters *both_pro* and *i_pro* are too large. That is to say that persistent multiple attacks can be successful with a certain probability, but the probability is generally lower than that of an unsuccessful attack. Figure 6 shows the probability density distribution of the two time-bounded reachability properties, the abscissa means of which run for a duration in time. Therefore, we can conclude that in most cases, the probability of entering a safe state is higher than that of entering an unsafe state. In some cases, the probability of the two states is similar. Thus, the G2F framework defense mechanism is relatively weak against persistent multiple attacks, and caution is still needed.

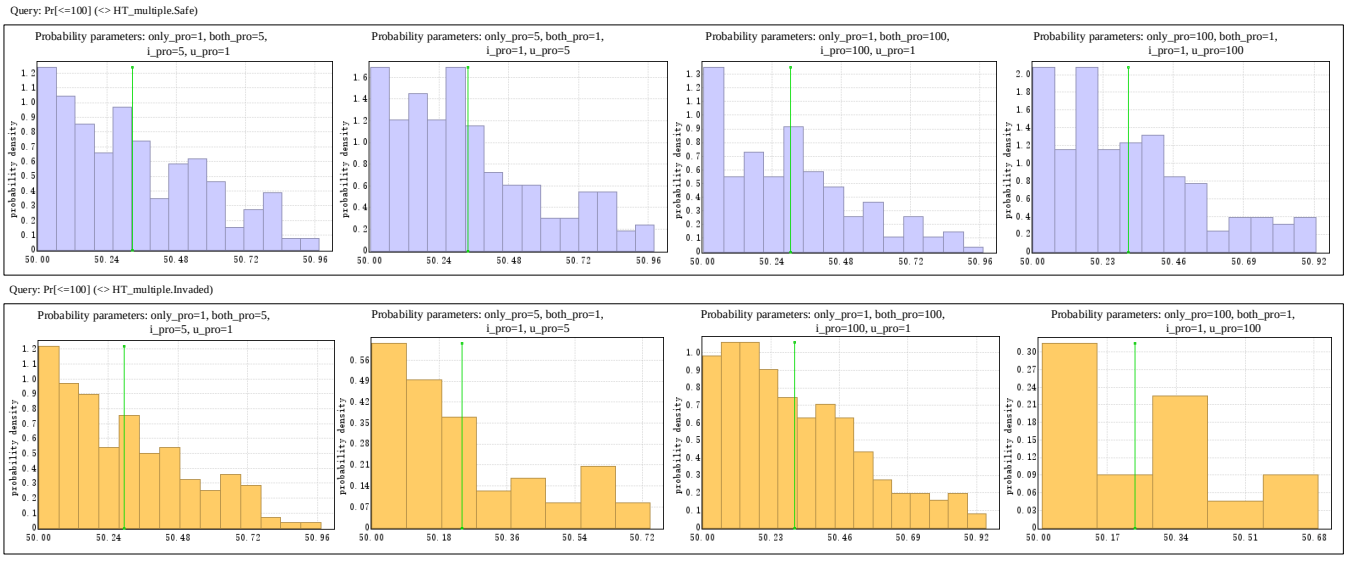


Figure 6. Statistics in four different probability settings for persistent multiple attacks when the user pushes the button at the 50th time unit and the intruder attacks every 5 time units.

Table 6. Properties in UPPAAL.

$Pr[<=100] <> (HT\_single.Safe)$	(1)
$Pr[<=100] <> (HT\_single.Invaded)$	(2)
$Pr[<=100] <> (HT\_multiple.Safe)$	(3)
$Pr[<=100] <> (HT\_multiple.Invaded)$	(4)

4.4. Proposed Algorithms

Algorithm 1 is designed to build an HLPSL model. We instantiate and explain the algorithm by building the HLPSL model of the hardware token in Table 2. Line 2 in Algorithm 1 provides the roles and their parameters (corresponding to line 1 in Table 2). Lines 3–5 in Table 2 generate their parameters and corresponding types (through line 9 of Algorithm 1). The transition starting from lines 9–12 in Table 2 is generated by dividing the functions of different entity roles (through lines 12–22 in Algorithm 1).

Algorithm 2 supports the construction of CSP models for entities. Here, we demonstrate the process of constructing a partial CSP model for the hardware token. Specifically, we refer to steps 5–7 and extract three messages to form a message set (line 1 in Algorithm 2). Based on the local known elements of the hardware token, which include *counter* and *signature*, we can obtain line 1 in Table 3 (through line 1 in Algorithm 2). Taking the first message (corresponding to step 5) as an illustration, we identify it as a transferring action and thus find channel *ComGH_U*, along with the corresponding message ({*handle*, *app_id*, *challenge*, *origin*, *channel_id*}) (through lines 3–7 in Algorithm 2). Additionally, since it is a sending action, the " ′ " suffix is added to the element of the message, resulting in lines 1–2 of Table 3 (through lines 8–10 in Algorithm 2).

Algorithm 3 is capable of converting a CSP model into a Timed CSP model. Taking the hardware token as an example, the CSP model (*HT_Auth*()) (shown in Table 3) is first divided into a process head (*ph*) (line 1 in Table 3 without "$=_{df}$") and a process body (*pb*) (lines 2–4 in Table 3), which is achieved by line 1 in Algorithm 3. The *pb* does not contain any operations that need to be executed within a certain time, but it does require a waiting time (t_2) before execution. Therefore, *WAIT* t_2 needs to be added to the beginning of the *pd*, and *ph* and *pb* are combined again to obtain the Timed CSP model (*tp*) (lines 2–10 in Algorithm 3), which is shown in lines 5–8 in Table 3.

4.5. Discussion

Currently, MFF-IoT only supports the modeling of operations for general authentication (e.g., encryption and decryption), but it can easily be extended to support other operations using C# integrated into PAT. MFF-IoT is only able to detect signature-faking attacks, single attacks, and persistent multiple attacks; it can also easily be improved by building more formal models for other attack types. Algorithms 1–3 proposed in our framework are reusable and support the modeling of new authentication schemes. They facilitate security analysis in the coarse-grained formal verification and the first fine-grained formal verification. Moreover, this methodology can only be used for user authentication in IoT applications, which means that it does not consider other IoT applications. Although the time to complete each verification is less than 2 s in our case study, an increase in the time parameter may lead to a time increase for fine-grained verification. We will keep improving our methodology to solve these issues.

5. Conclusions and Future Work

In this paper, we have provided a multi-granularity formal framework of user authentication for IoT (MFF-IoT) that includes one coarse-grained modeling and two fine-grained modelings. We used G2F as a case study and formalized it with the AVISPA tool to verify fundamental security properties as a coarse-grained formal verification. We applied process algebra CSP, Timed CSP, and timed automata to build three formal models, then fed them to PAT supported by C# and UPPAAL, which completed two fine-grained formal verifications with gradually finer granularity. We also verified three security properties

(deadlock freedom, signature verification, and intruder success), as well as security probability properties, with observable statistics. Based on the verification results, the ability of G2F to resist hardware token faking attacks is influenced by the frequency with which such attacks occur. We represent attack frequency as different weights on the edges of the timed automata, and as this frequency increases, the likelihood of worse system performance also increases. We intend for our MFF-IoT to assist in analyzing the performance of authentication mechanisms in a vulnerable environment in which hardware token faking attacks are possible.

In addition to the vulnerability posed by hardware token faking attacks, there may be other types of attacks that create vulnerable environments, which could be considered in future work. We plan to prevent attacks for the case study based on our verification results and explore strategies to extend the applicability of MFF-IoT to other IoT applications that require user authentication. To improve the usability and reusability of MFF-IoT, we intend to implement algorithms that make the conversion process more automatic and efficient. We also aim to integrate reliability metrics into our framework to provide users with a more comprehensive understanding of their system's security.

In summary, our future work includes preventing attacks for the case study, extending the applicability of MFF-IoT to other IoT applications, improving the usability and reusability of MFF-IoT, and integrating reliability metrics into our framework.

Author Contributions: Conceptualization, Y.F.; Validation, Y.F.; Formal analysis, Y.F., J.Y. and L.Y.; Visualization, J.Y.; Supervision, L.Y. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Fundamental Research Funds for the Central Universities, Shanghai Key Laboratory of Trustworthy Computing (Grant No. OP202003) and 2023 Shanghai Educational Science Research Program (Grant No. C2023039).

Data Availability Statement: No new data were created or analyzed in this study. Data sharing is not applicable to this article.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Dragoni, N.; Giaretta, A.; Mazzara, M. The Internet of Hackable Things. *arXiv* **2017**, arXiv:1707.08380.
2. Spognardi, A.; Donno, M.D.; Dragoni, N.; Giaretta, A. Analysis of DDoS-Capable IoT Malwares. In Proceedings of the 2017 Federated Conference on Computer Science and Information Systems, Prague, Czech Republic, 3–6 September 2017; pp. 807–816.
3. Ouaddah, A.; Kalam, A.A.E.; Ouahman, A.A. FairAccess: A new Blockchain-based access control framework for the Internet of Things. *Secur. Commun. Netw.* **2016**, *9*, 5943–5964. [CrossRef]
4. Aktas, M.S.; Astekin, M. Provenance aware run-time verification of things for self-healing Internet of Things applications. *Concurr. Comput. Pract. Exp.* **2019**, *31*, e4263. [CrossRef]
5. Shkarupylo, V.; Kudermetov, R.; Timenko, A.; Polska, O. On the Aspects of IoT Protocols Specification and Verification. In Proceedings of the 2019 IEEE International Scientific-Practical Conference Problems of Infocommunications, Science and Technology (PIC S&T), Kyiv, Ukraine, 8–11 October 2019; pp. 93–96.
6. Drozdov, D.; Patil, S.; Dubinin, V.; Vyatkin, V. Towards formal verification for cyber-physically agnostic software: A case study. In Proceedings of the IECON 2017—43rd Annual Conference of the IEEE Industrial Electronics Society, Beijing, China, 9 October–1 November 2017; pp. 5509–5514.
7. Hameed, K.; Khan, A.; Ahmed, M.; Alavalapati, G.R.; Rathore, M.M. Towards a formally verified zero watermarking scheme for data integrity in the Internet of Things based-wireless sensor networks. *Future Gener. Comput. Syst.* **2018**, *82*, 274–289. [CrossRef]
8. Viganò, L. Automated Security Protocol Analysis With the AVISPA Tool. In Proceedings of the 21st Annual Conference on Mathematical Foundations of Programming Semantics, MFPS 2005, Birmingham, UK, 18–21 May 2005; pp. 61–86.
9. Liu, Y.; Sun, J.; Dong, J. An Analyzer for Extended Compositional Process Algebras. In Proceedings of the Companion of the 30th International Conference on Software Engineering, New York, NY, USA, 10–18 May 2008; pp. 919–920.
10. Larsen, K.G.; Pettersson, P.; Yi, W. UPPAAL in a Nutshell. *Int. J. Softw. Tools Technol. Transf.* **1997**, *1*, 134–152. [CrossRef]
11. Chevalier, Y.; Compagna, L.; Cuellar, J.; Drielsma, P.H.; Mantovani, J.; Moedersheim, S.; Vigneron, L. A High Level Protocol Specification Language for Industrial Security-Sensitive Protocols. In Proceedings of the Workshop on Specification and Automated Processing of Security Requirements (SAPS'2004), Linz, Austria, 20–25 September 2004. Available online: https://hal.inria.fr/inria-00099882/(accessed on 28 March 2023).

12. Luo, H.; Wang, C.; Luo, H.; Zhang, F.; Lin, F.; Xu, G. G2F: A Secure User Authentication for Rapid Smart Home IoT Management. *IEEE Internet Things J.* **2021**, *8*, 10884–10895. [CrossRef]
13. Hofer-Schmitz, K.; Stojanovic, B. Towards formal verification of IoT protocols: A Review. *Comput. Netw.* **2020**, *174*, 107233. [CrossRef]
14. Aziz, B. A formal model and analysis of an IoT protocol. *Ad Hoc Netw.* **2016**, *36*, 49–57. [CrossRef]
15. Mohsin, M.; Anwar, Z.; Husari, G.; Al-Shaer, E.; Rahman, M.A. IoTSAT: A formal framework for security analysis of the internet of things (IoT). In Proceedings of the IEEE Conference on Communications and Network Security, Philadelphia, PA, USA, 17–19 October 2016; pp. 180–188.
16. Mahadewa, K.T.; Wang, K.; Bai, G.; Shi, L.; Dong, J.S.; Liang, Z. HOMESCAN: Scrutinizing Implementations of Smart Home Integrations. In Proceedings of the 23rd International Conference on Engineering of Complex Computer Systems, Melbourne, VIC, Australia, 12–14 December 2018; pp. 21–30.
17. Mahadewa, K.; Wang, K.; Bai, G.; Shi, L.; Liu, Y.; Dong, J.S.; Liang, Z. Scrutinizing Implementations of Smart Home Integrations. *IEEE Trans. Softw. Eng.* **2021**, *47*, 2667–2683. [CrossRef]
18. Brookes, S.D.; Hoare, C.A.R.; Roscoe, A.W. A Theory of Communicating Sequential Processes. *J. ACM* **1984**, *31*, 560–599. [CrossRef]
19. Hoare, C.A.R. *Communicating Sequential Processes*; Prentice Hall: Upper Saddle River, NJ, USA, 1985.
20. Roscoe, A.W. *The Theory and Practice of Concurrency*; Prentice Hall: Upper Saddle River, NJ, USA, 1997.
21. Roscoe, A.W. *Understanding Concurrent Systems*; Texts in Computer Science; Springer: Berlin/Heidelberg, Germany, 2010.
22. Davies, J.; Schneider, S.A. A Brief History of Timed CSP. *Theor. Comput. Sci.* **1995**, *138*, 243–271. [CrossRef]
23. Mir, O.; van der Weide, T.P.; Lee, C. A Secure User Anonymity and Authentication Scheme Using AVISPA for Telecare Medical Information Systems. *J. Med. Syst.* **2015**, *39*, 89:1–89:16. [CrossRef] [PubMed]
24. Si, Y.; Sun, J.; Liu, Y.; Dong, J.S.; Pang, J.; Zhang, S.J.; Yang, X. Model checking with fairness assumptions using PAT. *Front. Comput. Sci.* **2014**, *8*, 1–16. [CrossRef]
25. Asokan, S.; Kumar, G.S. Modelling and Verification of the FlexRay Startup Mechanism using UPPAAL Model Checker. In Proceedings of the 2018 8th International Symposium on Embedded Computing and System Design (ISED), Cochin, India, 13–15 December 2018; pp. 69–73.
26. Ravn, A.P.; Srba, J.; Vighio, S. A Formal Analysis of the Web Services Atomic Transaction Protocol with UPPAAL. In Proceedings of the ISoLA 2010, Heraklion, Greece, 18–21 October 2010; pp. 579–593.
27. Masud, M.; Gaba, G.S.; Choudhary, K.; Hossain, M.S.; Alhamid, M.F.; Muhammad, G. Lightweight and Anonymity-Preserving User Authentication Scheme for IoT-Based Healthcare. *IEEE Internet Things J.* **2022**, *9*, 2649–2656. [CrossRef]