Formal verification of the FDO protocol

Formal Verification of the FDO Protocol

Simone Bussa, Riccardo Sisto, Fulvio Valenza
Dip. Automatica e Informatica, Politecnico di Torino, Torino, Italy, Emails: {first.last}@polito.it

***Abstract*—Fido Device Onboarding (FDO) by the FIDO Alliance is expected to become the leading standard protocol for device secure onboarding. It is automatic, plug&play and it enables late binding. It overcomes the problems related to both proprietary zero-touch protocols and the traditional manual approach. Being automatic, this protocol must enforce a large number of security properties. In this paper, for the first time, we formally verify whether it satisfies some of its expected properties. Authentication and confidentiality of exchanged secrets are checked using a symbolic model and the Proverif tool. The verification confirms a weakness in the way the transfer of the device ownership is handled, which was already guessed by other authors but never described in detail.**

***Index Terms*—Device onboarding, FIDO FDO, Security analysis, Formal Verification**

I. Introduction

Device onboarding, also called "device provisioning", is the process by which a newly unboxed device connects for the first time to the platform on which it will operate. Usually, the devices to be onboarded are IoT devices with limited computational and storage capabilities, which must be connected to a cloud or "Platform" with, on the contrary, high resource potential.

The onboarding process today is still mostly done manually: an operator turns on the device, logs into its terminal, and configures all the credentials and information needed to find the platform on the network to connect to. This type of approach has several weaknesses. (i) It is a lengthy process; it takes a lot of time to perform it manually, especially when large "swarms" of devices need to be onboarded. For this reason, it is not scalable. (ii) It is necessary to trust the operator who carries out the operations, in addition to the fact that this person must be qualified and have experience in the field. (iii) The costs of the process are consequently high.

One solution being sought is to automate the process. Automation also brings with it a large number of challenges. The heterogeneity of IoT devices, each with its own hardware and operating system, and the presence of different vendors in the field are just a few examples. This has given rise to several Proprietary Zero Touch protocols, that are automated but usable only with specific platforms from a given vendor. Their drawback is that they require the device to be programmed already during production to communicate with certain vendors and platforms, thus severely limiting flexibility and portability.

In 2020, The FIDO Alliance, backed by most of the global tech leaders, proposed a draft of an open standard for automatic device onboarding, called Fido Device Onboarding (FDO, [1]), which aims to reconcile all existing proprietary protocols under one umbrella. FDO is hardware-independent, plug&play, and allows late binding. With this latter feature, it is possible to choose the platform on which to onboard the device at any point during its lifecycle, without having to specify this information at the manufacturing stage. The process is designed so that, when the device is turned on, it can find the platform to connect to by itself and autonomously perform all the necessary operations without requiring any manual intervention.

Since humans have no control over the process, this kind of automatic protocol must satisfy a large number of security and privacy properties. Indeed, mutual authentication between the parties and the integrity of the messages exchanged must be guaranteed. An attacker must not be able to participate in the protocol and pretend to be a valid honest entity. Moreover, the privacy of the device must also be ensured. The final destination of the device (i.e., the platform it will be onboarded on) shall not be exposed, as well as subsequent calls to the protocol must not allow linking past and future owners of the device. In the first standard document published in 2020 [1], and in the subsequent 2022 version [2], FIDO did not specify a concrete threat model for its process. However, it is reasonable to assume that it follows the same threat models as existing similar device onboarding protocols since the context in which FDO works is the same.

FDO is expected to become the leading standard for automatic device onboading. For this reason it is necessary to deeply analyze its security and verify that the expected properties are actually met. One of the ways this can be achieved is through formal verification, which, when applied to network protocols, allows to reach the highest guarantees of the security of the analyzed protocol among the various existing static analysis methods. Unlike other FIDO protocols that were standardized years ago and for which there are numerous security verifications conducted over the years, for FDO there seems to be no formal verification study in the literature. The objective of this paper was therefore to perform a first formal analysis of the protocol to highlight any present vulnerabilities and thus have more assurance about its security.

The remainder of this paper is structured as follows. In Section II, we summarize some related work. Section III gives a detailed description of the FDO protocol. Section IV presents the model used for the formal verification. Finally, Section V discusses the verification results, and Section VI the conclusions.

II. Related Work

Several papers have been published over the years analyzing the security of IoT devices. Some examples are [3], [4], [5]. [3], [4] analyze the limitations of IoT devices (e.g., limited computational resources, low battery, etc.) and what solutions have been adopted to overcome these problems in terms of security and privacy (e.g., lightweight encryption scheme). They both refer to existing IoT frameworks and schemes proposed in the literature, and describe how they handle authentication, authorization, and access control. [5] is a framework that provides examples of architectures and best practices for building secure and reliable IoT systems. It is a collaboration between various industrial IoT company leaders aiming to provide a concrete guide for anyone interested in building an IoT environment, with a strong focus on its safety and security.

However, the papers described above analyze the security of the entire IoT device lifecycle, without focusing much on the onboarding phase. The concept of automatic onboarding is more recent. Initially, in fact, it was carried out manually by a system administrator, and thus it was not a subject of study for possible network security attacks. The need to automate the process has brought to light several new protocols. Although several groups have tried to propose their automatic solution, there is still no shared accepted standard. Starting in 2020, the National Institute of Standards and Technology (NIST) has launched a project to try to address this gap, [6]. To date, the project is still a work in progress.

[7] is a white paper published by the Industry IoT Consortium (IIC) that exhaustively summarizes all published protocols to perform automatic secure onboarding, updated to October 2022. It describes how these protocols handle security and privacy and what infrastructure (e.g., PKI) is needed to assist the process. Among the protocols described, the FIDO FDO [1] is presented, based on the Intel Secure Device Onboarding (SDO) [8].

Regarding the FDO protocol, in particular, being a recent publication, we did not find any work in the literature that analyzes its security. We only found a paper, [9], that critically analyzes and questions the excessive trust that FDO assumes towards the supply chain. Driven by these motivations, we decided to formally analyze the FDO protocol.

III. FIDO Device Onboarding

This section shows in detail how the FDO protocol for secure device onboarding works. The protocol is described in the standard document [1], and in its subsequent version [2]. The objective of FDO, besides connecting the device to the platform on which it will operate, is three-fold:

1) Authentication of the device to the platform (also called the *Owner Platform*, or simply the *Owner*)
2) Authentication of the platform to the device
3) Exchange of cryptographic material to generate a symmetric session key, to be used to encrypt and protect all communications between the device and the owner platform, once the onboarding protocol has ended.

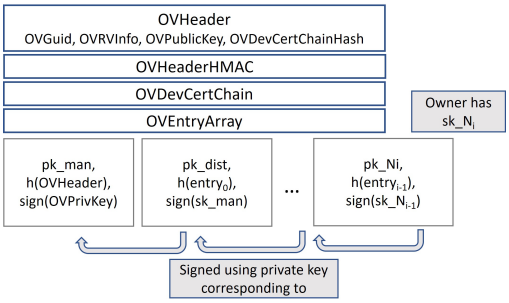

Fig. 1: Ownership Voucher structure

The onboarding process must be automatic: the device, once turned on, must be able to find the IP of the owner platform autonomously, without any manual intervention (and without this information being configured in the device during manufacturing). For this purpose, it is necessary to introduce an additional entity, the *Rendezvous Server*. It is a third party, which acts as a meeting point between the Device and the Owner. The device is configured during manufacturing to communicate upon power-up with the Rendezvous Server. Before the device is turned on, the Owner contacts the server, authenticates itself as the legitimate owner of the device, and stores its URL (Owner URL) on the server. Then, it starts waiting for a connection at the specified URL. Once turned on, the device contacts the server to retrieve the URL of the owner to connect to.

To prove the ownership of the device, the Owner uses the Ownership Voucher (*OV*). This is a particular certificate whose (simplified) structure is described below and shown in Fig. 1.

- $OVDevCertChain$. It contains the entire certification chain of the device. Therefore, the public key of the device is certified by a list of CAs, rooted by a Root CA trusted by all the entities involved in the protocol.
- $OVHeader$. It contains the device unique identifier, $OVGuid$, the IP address of the server the device has to contact at power-up, $OVRVInfo$, the public key of the manufacturer that built the device, $OVPublicKey$, and the hash computed on the $OVDevCertChain$.
- $OVHeaderHMAC$. It is an HMAC computed by the device on the $OVHeader$, combined with a $secret$ generated by the device and saved internally. Only the device, which has the secret, can perform this computation.
- $OVEntryArray$. It is an array of $Entry$ that identifies the current and previous owners of the device. Whenever the owner changes, a new entry is added to the array.

Each $OVEntry$, except the first one, contains a public key, a hash of the previous entry, and a signature of the previous fields. Initially, the device is owned by the manufacturer, so the first entry includes pk_{man}, which is equal to $OVPublicKey$ contained in the header. Instead of the hash of the previous entry, the first entry contains the hash of the OVHeader. The first entry is signed using the private key of the manufacturer. Each time the device is sold, a new entry is added, containing

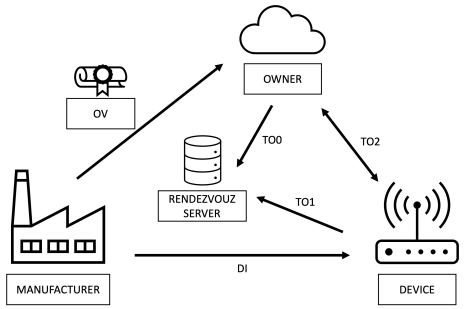

Fig. 2: FDO General Scheme

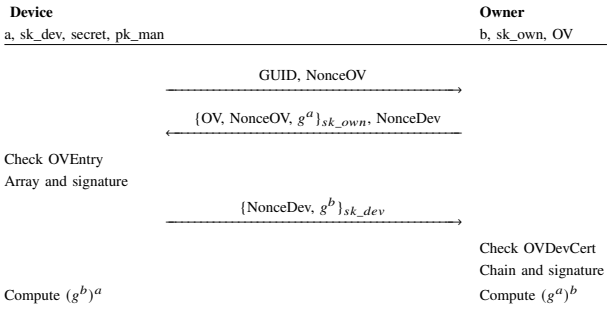

Fig. 3: FDO Transfer Ownership 2 (TO2)

the public key of the new owner (e.g., the public key of the distributor, pk_{distr}), the hash of the previous entry, and a signature of the previous fields, done using the private key of the previous owner, that is the key associated to the public key contained in the old last entry.

Let us now describe the protocol. The general scheme is shown in Fig. 2. It consists of four phases.

1) Device Initialize (DI). This phase takes place inside the manufacturer's factory (i.e., in a safe environment). The device is created and configured with a unique GUID and a private-public key pair, for which the manufacturer has obtained a certificate from the CA, DevCertChain. In this phase of the protocol, the Manufacturer creates the OV. First of all, it chooses the Rendezvous Server to use: there may be some sort of agreement between the two. It then builds the OVHeader with all the necessary fields and sends it to the device. The device, in turn, creates a secret and computes the HMAC of the received OVHeader using the secret as the key. This HeaderHMAC is returned to the Manufacturer, which inserts it into the OV. The device internally maintains the secret and the manufacturer public key (i.e., OVPublicKey taken from the OVHeader). Both will be used in the later stages of the protocol.

2) Transfer Ownership 0 (TO0). Phase that registers the Owner on the Rendezvous Server. The owner has purchased the device and obtained the OV. The Owner public key is contained in the last entry of the OVEntryArray. It was inserted by the previous intermediate node in the supply chain, when selling the device to the Owner. From the OVHeader, the Owner extracts the IP address of the server to contact, and contacts it. Upon request, the server sends a nonce to the Owner to request proof of ownership of the OV. The Owner replies by sending the OV, the nonce, and its own URL, all signed using the owner private key. The server verifies the signature on the response message and checks the integrity of the OV. Specifically, it sequentially verifies each OVEntry, checking its signature (using the public key contained in the previous entry) and hash (which must match the hash of the previous entry). By checking the first entry, the server can also verify the integrity of the OVHeader (because the first entry contains the hash of the header). The server uses OVPublicKey as the root of trust, so it is assumed that the server at least trusts the manufacturer public key. We can think that, since the manufacturer chooses the server, there is some sort of agreement between the two. Once the owner has been successfully authenticated, the server stores the pair device GUID (and OV) - Owner URL, to be returned to the device in a later stage. The owner, on the other hand, starts waiting for a connection from the device on the specified URL.

3) Transfer Ownership 1 (TO1). The device contacts the Rendezvous Server to obtain the IP address of the Owner. It sends its GUID and receives a nonce to sign to authenticate itself. Then, the device signs the nonce using its private key. The server retrieves the correct OV associated with the GUID of the device and verifies the signature on the nonce using the device public key contained in OVDevCertChain (it also verifies the entire certification chain of the device). If the device authentication is successful, the server returns the URL of the Owner.

4) Transfer Ownership 2 (TO2). This is the main phase of the FDO protocol. The device and the owner perform mutual authentication and exchange the symmetric session key. This phase is schematized in Fig. 3. The device contacts the Owner at the URL returned by the Server, and sends its GUID and a nonce, NonceOV, to be signed by the owner to demonstrate proof of possession of the OV. The owner replies by sending the OV, the received nonce, and the first parameter of the selected key exchange protocol, g^a. To perform key exchange, FDO supports several protocols. The owner and the device can negotiate which one to use. Here we focus on the case when the negotiated protocol is Diffie-Hellman, so the owner generates a secret a and computes g^a. The owner signs the entire message using its private key, sk_own, whose corresponding public key is contained in the last entry of the OV. Along with the message, it also sends a nonce, NonceDEV, which the device must sign to authenticate itself. The device receives the message and first verifies the OVEntryArray: it checks the signature in each entry, whether the last entry contains the public key of the owner, and whether the first one contains the public key of the Manufacturer (the device saved this key internally during DI protocol). Then it also checks the integrity of the OVHeader by recomputing the HMAC using the secret it has stored internally. Finally, it verifies the signature on the received message using the public

key of the Owner. If all the checks are ok, the device replies to the Owner sending NonceDev and the second parameter for the key exchange protocol, g^b, signed using the device secret key. The Owner verifies the signature using the device public key contained in the OVDevCert. At this point, the Owner and Device have completed mutual authentication and compute the Diffie-Hellman symmetric session key as $g^{ab} = g^{ba}$. In this case, the executed key exchange protocol is Authenticated Diffie-Hellman. The shared symmetric key will be used to secure all subsequent communications.

IV. Formal Analysis

This section describes how the FDO protocol was formally verified. Formal verification is a type of static analysis that allows to rigorously analyze the security of a protocol by constructing a formal model of it and by checking that the model satisfies some security properties that must be enforced by the scheme. For the verification, automatic tools can be used. The one we adopted for the work in this paper is Proverif, [10]. It is a tool that takes as input a symbolic (abstract) model of the protocol and a set of properties also written in a formal way and automatically returns whether the properties are met by the protocol under certain reasonable modeling assumptions or whether some attacks are possible. The tool is sound but not complete: this means that sometimes the tool can return false attacks, but if it proves the correctness of a property, the property is guaranteed to hold on the model.

The most difficult part of formal verification is the modeling part, that is, translating the actual protocol into an abstract model capable of representing it. In the case of Proverif, the model is symbolic. The main difference with other formal models (e.g., computational ones) is that symbolic models assume perfect cryptography: cryptographic operations are black box functions, modeled through function symbols in an algebra of terms. It means they are assumed to be unforgeable and all attacks that exploit their weaknesses are not detected in the verification. However, the verification can find attacks enabled by logical flows (e.g., man-in-the-middle attacks). The cryptographic functions included in the FDO protocol are Digital Signatures, Symmetric Encryption, Hash functions, HMAC, and Diffie-Hellman Exponentiation, all modeled through standard equations that can be found on the Proverif manual, [10].

In the following, we discuss the various aspects of the model used for the formal verification in Proverif[1]. The model was verified on a machine with an i7-6700 CPU with 32 GB of RAM. The verification results all return within 30 minutes after the start of the execution.

A. Component model

All the main actors involved in the FDO protocol were also represented in the Proverif model. Thus, there are (i) Devices with a protected memory and a public-private key pair certified by a CA, (ii) Manufacturer that builds devices and has an agreement with the Rendezvous Server, (iii) Rendezvous Server chosen by the Manufacturer, (iv) Owner which wants to onboard the device. The model assumes there can be multiple devices but only one manufacturer, owner, and server. However, each of these entities can perform multiple runs of the protocol. E.g., the Manufacturer can build several devices, the server can accept multiple connections, and the owner can onboard different devices.

Of great importance is also the OV, which is modeled as a certificate containing multiple entries in a chain. To reduce the complexity of the formal verification, we put a limit on the number of entries that can be simultaneously present in the OV and we set this value to 3. This means that the device can change a maximum of 3 owners before being onboarded. We believe that this assumption is reasonable and does not limit the security analysis conducted on the protocol, allowing at the same time Proverif to terminate in a reasonable time without requiring excessive computational resources.

B. Channel and Threat model

All communication channels between the various entities are modeled as public channels, except for the channel between the device and manufacturer, which is private. In fact, it is assumed that the DI protocol, used to initially configure the device, takes place in the manufacturer's factory and therefore in a safe environment. On the other hand, the other communications are all potentially insecure. On public channels, the attacker behaves as a classic Dolev-Yao attacker: it can intercept, read, modify, and replay messages. The attacker can also perform cryptographic operations using the keys in its possession. However, it cannot perform brute force operations (e.g., reverse a hash function or decrypt a message if it does not know the decryption key).

We performed formal verification of the protocol in different scenarios, corresponding to different threat models.

First scenario. In this first case, the attacker is not involved in the protocol as a legitimate entity. This means it tries to hack the process from the outside as an external entity. It can use the messages exchanged between the various actors to increase its knowledge and try to break the protocol. This is the most common case, in which both the supply chain and the owners are all trusted.

Second scenario. In this second case, the attacker owns a second valid device issued by the manufacturer with a GUID, $GUID_{att}$, different from the one of the honest devices, $GUID_x$. In this scenario, the attacker can participate in all phases of the protocol as a legitimate entity, being the owner of a valid device that can be onboarded. Therefore, it has a valid OV released by the manufacturer associated with $GUID_{att}$ and can correctly authenticate to the Rendezvous Server and store its URL. The underlying threat model of this second scenario is to assume that owners may be dishonest. In this case, the model lets us check, for example, whether there exists a strict correlation between a device and its OV/credentials. In other words, if this correlation exists, the attacker should not be able to use the credentials/OV of the device with $GUID_{att}$

[1]The Proverif source code can be found at this link https://github.com/netgroup-polito/verification-fdo

to execute the protocol in place of a device with $GUID_x$. For example, it should not succeed in storing its URL_{att} associated with $GUID_x$ on the server, despite being able to authenticate correctly using the OV of $GUID_{att}$.

Third scenario. In this third case, the attacker is a dishonest intermediate node in the supply chain. It can purchase the device from the manufacturer (or any other intermediate node) and resell it later to another entity. In this way, it would have its own public key inserted among the entries in $OVEntryArray$ and use the corresponding private key to certify the transfer of ownership to another node (e.g., to the owner). This threat model is motivated by what was observed in [9], which claimed the FDO protocol does not have adequate protection against untrusted actors in the supply chain.

C. Security properties

The properties verified using Proverif are *confidentiality* and *authentication*.

Specifically, we tested the secrecy of the symmetric key shared between the device and the owner at the end of the key exchange protocol in the TO2 phase. If the attacker possessed this key, he would be able to decipher all subsequent application messages exchanged at the end of the onboarding process.

Another property that was verified is authentication, both authentication of the Owner to the Device and, vice versa, of the Device to the Owner. Thus, we expect that only the Owner, as the unique legitimate owner of the OV, can authenticate correctly to the device using this certificate. On the contrary, the device is expected to be the only entity able to authenticate to the Owner using its private key corresponding to $OVDevCertChain$ contained in the OV.

V. Verification Results

The formal verification confirmed that, in the first two scenarios of the considered threat model, all security properties are correctly satisfied by the protocol. Proverif has in fact found a formal proof that these properties are true in the model. In the third scenario, however, when the attacker can act as an intermediate node in the supply chain, not all properties were successfully verified. This weakness related to the supply chain was already guessed by [9], in which the authors criticized the excessive trust the protocol assumes toward the supply chain, but a specific attack was not described.

In this scenario, the protocol verification revealed the following possible attack. Let us imagine that the attacker purchased the device from the Manufacturer and later sold it to the Owner. The resulting OV received by the Owner would be the one shown in the left in Fig. 4. In this case, the attacker has its public key, pk_att, inserted in the second OVEntry, correctly signed by the manufacturer. When selling the device to the Owner, the attacker adds a third entry containing the Owner public key signed with the attacker private key, sk_att.

What could happen is that the attacker keeps a copy of the OV with two entries (shown in the right in Fig. 4) and

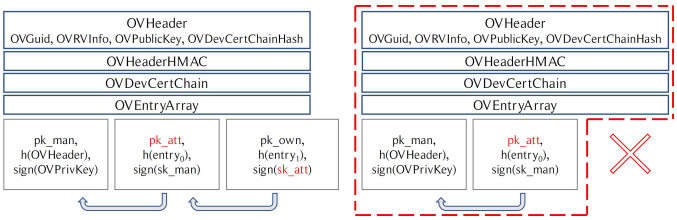

Fig. 4: OV 3 entries VS OV truncated 2 entries

uses this "truncated" version to authenticate as the rightful owner of the device, even after selling the device to the real Owner. In fact, in this case, both the Owner with the three-entries version of the OV and the attacker with the two-entries version can authenticate to the Device and Rendezvous Server as legitimate owners. This clearly violates the authentication property. This happens because the device is completely unaware of how many changes of ownership occurred while it was turned off, which could have been two or three.

This violation of the authentication property returned by Proverif could lead to the following attacks.

A. Denial of Service

The attacker could act in the TO0 phase of the protocol to authenticate to the Rendezvous Server as the correct owner of the device. In this case, it could register its URL associated with the GUID of the device. In the TO1 phase, when the device requests the URL of the owner to the server, the URL of the attacker would be returned. This would make the device unusable (Denial of Service) since it would never try to open a connection with the real owner.

Moreover, the FDO standard document specifies that in case TO0 is executed multiple times, each new request overwrites the previous ones, with the new association GUID-URL replacing the old one stored on the server. This is to give the owner the ability to change the URL on which to wait for a connection at any time. Because of this, it is not important for the attacker to contact the server before the Owner. If it contacted the server after the real Owner had already saved its URL, its new request would override that made by the real Owner. To solve this issue, the server should perform an additional check on the received OV used for authentication. For example, if it first authenticates the owner with a correct OV containing three entries, a subsequent request from the attacker with the same OV but truncated to two entries should be blocked. However, this does not solve the denial of service in case the attacker arrives first and blocks subsequent requests from the real Owner.

B. Attacker impersonating the Owner in TO2 phase

The same weakness that generated the attack above can also lead to a problem in the TO2 phase, where the attacker manages to authenticate to the device as its rightful owner. The attacker can show the device the truncated OV with two entries and claim to be its owner, acting as a man in the middle

between the device and the real Owner. The resulting attack is shown in Fig. 5.

The attacker can complete the key exchange protocol and thus share a symmetric key with the device. At this point, it would have full control of the device.

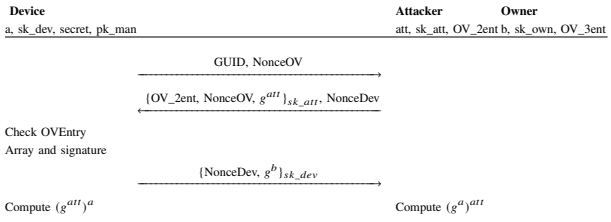

Fig. 5: Attacker authenticating to the device

C. Attacker impersonating the Device in TO2 phase

The authentication of the device to the Owner is more difficult to forge, as in the original protocol it is done using the device private key that is stored in a protected memory. Thus, the only way for the attacker to impersonate the device is by compromising the OV and replacing the device certificate with a certificate of which it knows the corresponding private key. But since the OVHeader contains the hash of the OVDevCertChain, the attacker must modify the header as well.

At this point, however, there would be an inconsistency with the first entry of the OVEntryArray, which contains the hash of the OVHeader. Changing even this field is impossible for the attacker because the entire OVEntry is signed with the manufacturer private key, which is not known to the attacker.

Here, there are two distinct possible derivations, both returned by Proverif.

- The attacker also modifies OVPublicKey, which is the manufacturer public key, contained in the header. In this field, it inserts its own public key and signs the first OVEntry with the corresponding private key. This attack is possible if the owner did not know the manufacturer public key (and thus it is easy for the attacker to replace it in the header with its own key). The resulting OV received by the Owner is the one shown in Fig. 6. With this OV, the attacker can authenticate as a valid device to the Owner, sending the message $\{NonceDev, g^{att}\}_{sk_{att\_cert\_sk}}$. However, we think this scenario is quite unrealistic.
- In this second scenario, the owner knows and trusts the public key of the manufacturer, so OVPublicKey cannot be substituted. As a result, even the first OVEntry, signed with the corresponding manufacturer private key, cannot be changed. In this case, the attacker would not be able to authenticate as a valid device to the Owner.

VI. Conclusions

In this paper, we have performed a first formal symbolic analysis of the FDO protocol draft proposed by the FIDO

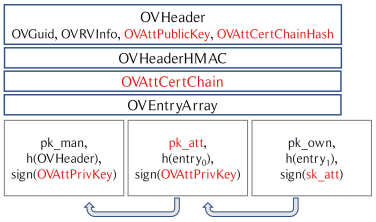

Fig. 6: Attacker authenticating to the Owner

Alliance for automatic secure device onboarding. The formal verification has been conducted using Proverif. The results confirmed that the secrecy and authentication properties expected for the protocol hold on the protocol model if we assume the supply chain is trusted. Such properties still hold, even under the assumption that some owners are not honest. Instead, the formal analysis confirmed the existence of a weakness, as already guessed in another paper in literature [9], when we assume the supply chain is not trusted. Our analysis found the specific attack that is possible in this case and that, up to our knowledge, has never been reported before. A malicious or compromised node within the supply chain could use a truncated version of the OV to pretend to be the rightful owner of the device. This could lead to the weaknesses described in this paper.

This work can help improve the protocol draft before it becomes a standard, used in the real world. It is important to remark that our analysis has been done on a model of the protocol and that it is still necessary to check whether the described attack works on a real implementation of the protocol. This check is left as future work.

References

[1] G. Cooper, B. Behm, A. Chakraborty, H. Kommalapati, G. Mandyam, H. T. ARM, and W. Bartsch, "Fido device onboard specification," 2020.

[2] ——, "Fido device onboard specification 1.1," *FIDO Device Onboard Specification*, vol. 1, 2021.

[3] M. Ammar, G. Russello, and B. Crispo, "Internet of things: A survey on the security of iot frameworks," *Journal of Information Security and Applications*, vol. 38, pp. 8–27, 2018.

[4] Y. Yang, L. Wu, G. Yin, L. Li, and H. Zhao, "A survey on security and privacy issues in internet-of-things," *IEEE Internet of things Journal*, vol. 4, no. 5, pp. 1250–1258, 2017.

[5] "Industrial internet of things security framework (IISF)," https://www.iiconsortium.org/IISF/, accessed: 2023-09-11.

[6] NIST project: Trusted internet of things device network-layer onboarding and lifecycle management. https://www.nccoe.nist.gov/projects/trusted-iot-device-network-layer-onboarding-and-lifecycle-management. Accessed: 2023-09-11.

[7] IIC automated onboarding and device provisioning: Best practices 2022. https://www.iiconsortium.org/iic-automated-onboarding-paper-form/. Accessed: 2023-09-11.

[8] "Intel secure device onboarding (SDO)," https://www.intel.com/content/www/us/en/internet-of-things/secure-device-onboard.html, accessed: 2023-09-11.

[9] K. Reaz and G. Wunder, "Asop: A sovereign and secure device onboarding protocol for cloud-based iot services," in *2022 6th Cyber Security in Networking Conference (CSNet)*. IEEE, 2022, pp. 1–3.

[10] B. Blanchet, B. Smyth, V. Cheval, and M. Sylvestre, "Proverif 2.00: automatic cryptographic protocol verifier, user manual and tutorial," *Version from*, pp. 05–16, 2018.