Formal Verification of the W3C Web Authentication Protocol

Iness Ben Guirat
INSAT
Tunis, Tunisia
inessbenguirat@gmail.com

Harry Halpin
Inria
Paris, France
harry.halpin@inria.fr

ABSTRACT

The science of security can be set on firm foundations via the formal verification of protocols. New protocols can have their design validated in a mechanized manner for security flaws, allowing protocol designs to be scientifically compared in a neutral manner. Given that these techniques have discovered critical flaws in protocols such as TLS 1.2 and are now being used to re-design protocols such as TLS 1.3, we demonstrate how formal verification can be used to analyze new protocols such as the W3C Web Authentication API. We model W3C Web Authentication with the formal verification language ProVerif, showing that the protocol itself is secure. However, we also stretch the boundaries of formal verification by trying to verify the privacy properties of W3C Web Authentication given in terms of the same origin policy. We use ProVerif to show that without further mandatory requirements in the specification, the claimed privacy properties do not hold. Next steps on how formal verification can be further integrated into standards and the further development of the privacy properties of W3C Web Authentication is outlined.

KEYWORDS

formal verification, ProVerif, authentication, W3C Web Authentication, science of security

ACM Reference Format:
Iness Ben Guirat and Harry Halpin. 2018. Formal Verification of the W3C Web Authentication Protocol. In *HoTSoS '18: Hot Topics in the Science of Security: Symposium and Bootcamp, April 10–11, 2018, Raleigh, NC, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3190619.3190640

For security to be a science, methodologies that can scientifically guarantee security and privacy properties such as formal verification need to be applied consistently to security issues. One of the largest problems damaging the security of modern information systems is the inability of users to generate and consistently use unique high-entropy passwords per domain. Fundamentally the entire idea of a password as symmetric secret for authentication is in need of replacement by asymmetric cryptography. In this paper, a proposed standard for public-key authentication, W3C Web Authentication, is analyzed using the formal verification. After describing the state of the art of formal verification in Section 1, we describe the background of the issue of authentication and the development of the W3C Web Authentication protocol in Section 2. In Section 3, each step of the W3C Web Authentication protocol is informally described. Using the formal verification tool ProVerif described in Section 4, we automatically formally verify the security properties of W3C Web Authentication in Section 5 and prove that the claimed privacy properties of W3C Web Authentication can be violated. A range of fixes outlined but not mandatory in the specification, such as the use of Direct Anonymous Authentication, are described in Section 6 and next steps for the scientific application of formal verification in terms of future protocols in Section 7.

1 THE SCIENCE OF SECURITY AND FORMAL VERIFICATION

Security has far too long been viewed as a black art based more on intuition than science, with protocols and cryptographic primitives often judged due to the reputation of their creator.[1] In contrast, by definition a science of security would place the security properties of techno-social systems on a scientific basis, including cryptographic primitives, the composition of those primitives into cryptographic protocols, and the embedding of these protocols into user behavior and the wider social world. This is a demanding task, and formal approaches have allowed tremendous amount of progress in the formal definition and verification of security properties [16]. Formalized game-hopping proofs have put the field of cryptography on a sound basis and can work over protocols of considerable complexity, but there exists still much work to be done to put this methodology into production for protocols. Although manual proofs for protocols do exist, even if specified in formal detail, complex protocols have large state spaces where a move in the protocol that was not foreseen by the designers may lead to the security properties of the entire protocol being violated. The classic example is that of the Needham-Shroeder protocol [22], whose security flaws were only discovered more than twenty years afterwards using formal methods [21].

A science demands a methodology and the methodology of formal verification tools provides a promising path for the verification of the security properties, and even privacy properties, of cryptographic primitives and protocols. Formal verification is the creation and checking of the properties of a cryptographic protocol or primitive not via proofs constructed by hand, but by proofs constructed in a fully mechanized manner. This is necessary as often the manual construction of proofs may, by sheer human accident, miss either seemingly routine aspects of the protocols that may

[1] For example, one may look at the CryptoForum Research Group debates between Daniel Bernstein and Microsoft Research that led to the fracturing of Microsoft and Google eco-systems in terms of elliptic curve support in TLS. See https://datatracker.ietf.org/doc/rfc7748/

HoTSoS '18, April 10–11, 2018, Raleigh, NC, USA

ACM ISBN 978-1-4503-6455-3/18/04…$15.00
https://doi.org/10.1145/3190619.3190640

violate their security and privacy properties, or that the level of complexity of the protocol is so such that the number of states that need to be inspected by hand are far greater than are possible within a reasonable amount of time. In modern protocols like TLS 1.2, it is precisely these kinds of obscure errors, such as the "triple handshake" renegotiation attack caused by complexity that were discovered via formal analysis [6].

While testing can help assess the security properties of a program, there will likely behavior of any program that is untested and can lead to the security properties being violated. Formal verification is the only approach that can guarantee correctness and security of a program. As formal verification via manual proofs are unlikely to cover all aspects of the security properties in every case, formal verification can be automated to explore the very large state spaces generated by programs. Note that this mechanized exploration is always within the constraints of the formalism used, so the entire state space may not be explored. Nonetheless, modern work in proof assistants such as *Coq* and the SMT solvers like *Z3* deployed by *F** [28] has progressed to the point where they can explore efficiently very large state-spaces that would otherwise be impossible via manual proofs.

Progress has accelerated recently. Tools such as ProVerif can model protocols for security properties such as the leakage of secret under the symbolic model [9], while new tools such as CryptoVerif allow even the automation of proofs under the computational model used in game-hopping proofs [7]. Two large advances have been the ability to test for *secret independence* by tools like *ct-verif*, i.e. constant-time verification, where it can be proven that the information flow and therefore implementation time is constant regardless of any secret material [3]. Another large advance has been the verified compilation of formally verified code in a higher-level language such as *F** into a safe subset of C, thus allowing formally verified code to actually be run in real-life environments [25], which is useful is one wants to create a new implementation of a protocol. If one wants to formally model an abstract standard like W3C Web Authentication that already has multiple implementations, ProVerif would be the tool of choice, as the browser vendors have already implemented the protocol.

However, the main problem is that these tools are typically deployed to verify implementations of standards of cryptographic primitives (NIST) or protocols (IETF, W3C, IEEE, ISO) after the standard is complete. In general, the methodology of formal verification would be more productively applied during the development of a new protocol itself, so that the standard's specification is assured to fulfill its security and privacy requirements. There has been work on API analysis like the W3C Web Cryptography API [10], but this analysis was done after the standard itself was completed, although efforts in TLS 1.3 have incorporated formal verification into the standardization process [5].

2 BACKGROUND

One new standard that is still being completed but has not yet been subjected to formal verification is W3C Web Authentication. The W3C Web Authentication protocol was created in order to get rid of passwords on the Web. Previous studies [13] and a large number of password database breaches have demonstrated that passwords (which are equivalent to a symmetric secret shared between the server and client) tend to be low-entropy and difficult, if not impossible to secure due to the increasing amount of power being deployed by brute-force attacks.[2] This has long been recognized as an outstanding problem and there has been a large array of previous research, ranging from password authenticated key exchange (PAKE) and smartcard-based solutions, but they have all failed to reach large market penetration due to patents and the inherent issues in deploying new technology to millions of users.[3] Therefore, researchers have long pointed out that a standard API for authentication across all web browsers would be one possible solution [18].

As the result of the W3C Workshop on *Web Cryptography Next Steps: W3C Workshop on Authentication, Hardware Tokens, and Beyond* in 2014, large vendors such as Paypal and all major browser vendors (Google, Microsoft, Mozilla)[4] have started work on a new standard for authentication at the World Wide Web Consortium (W3C).[5] The W3C is the world's leading Web standards body, responsible for standards like HTML and cryptographic standards such as the W3C Web Cryptography API that allow the access of cryptographic primitives in the Javascript runtime environment.

The core concept of W3C Web Authentication is to use asymmetric key material on user's devices to authenticate a user to a web server rather than passwords [30]. W3C Web Authentication (sometimes called "WebAuth" or "WebAuthn") is implemented by building a Javascript protocol that can take advantage of key material on any device, called the *authenticator*, in a generic manner. Authenticators can range from key material on hardware keys such as Yubikeys or key material in the secure enclave of smartphones. Not only does this allow for the authentication secret (the private key) to remain protected on the client device, it also means that this secret is unknown to the user and therefore cannot be stolen through phishing. A user with a device that supports W3C Web Authentication can register directly with key material using *one factor* cryptographic authentication. Cryptographic material can be used as a factor in *multi-factor* authentication (in addition to traditional password-based authentication or other factors like SMS and phone calls).

The concept behind W3C Web Authentication was first articulated in PhoneAuth [12], a framework that ensures user authentication by opportunistically providing cryptographic identity assertions from a user's mobile phone while the user authenticated using passwords with another device, such as their laptop. Previous standards around two-factor authentication called *Universal 2nd Factor* (U2F) [27] and *Universal Authentication Framework* (UAF) for one-factor cryptographic authentication were developed by FIDO (Fast IDentity Online) [12], a non-profit formed in 2012 with 250 corporate members. The FIDO UAF specifications not only saved users from using passwords, but specified recovery procedures when they forget a password as well as the integration of personal identification information such as biometric data stored locally on the user's device, as the local storage of biometrics and other personal information is intended to facilitate the concerns of

[2] Onlinehashcrack.com
[3] See US Patent *CN105721153 A*.
[4] Except Apple currently, although Apple often does not participate in the creation of new W3C standards.
[5] https://www.w3.org/2012/webcrypto/webcrypto-next-workshop/

users regarding personal data stored on an external server on the cloud. U2F was shipped with USB tokens deployed by companies internally such as Google and Github, and UAF with smartphones such as the Samsung Android.

The UAF and U2F specifications were simplified and unified in the FIDO 2.0 specification [29] that removed all references to specific factors such as biometrics and instead developed a unified API for the development that could deploy any number of authentication factors. The older FIDO protocol was formally verified using ProVerif, and it was detected that a missing check for an *AppID* could lead to attacks [23]. The *AppID* was removed in W3C Web Authentication and replaced by an *RP ID* whose checking is enforced. The FIDO 2.0 specification was submitted as W3C Member Submission to the W3C in 2015, and the W3C subsequently formed a W3C Working Group in 2016 to standardize the W3C Web Authentication API across all browsers. Google already accepts consumer-facing FIDO 1.0 and W3C Web Authentication using Yubikeys and other supported USB hardware tokens, as shown in Figure 1. A number of large changes from the FIDO 2.0 specification have happened, such as the simplification of the registration and authentication protocol, the enforcement of the same origin policy for privacy [26], and the use of asynchronous Javascript. Another related specification being developed at the IETF (Internet Engineering Task Force) in parallel to W3C Web Authentication API [4] is HTTPS Token Binding. The IETF HTTPS Token Binding standard is separate from W3C Web Authentication, although it is used to provide a TLS identifier for W3C Web Authentication if the client supports it.

Figure 1: Google Two-Factor Authentication Support

3 THE W3C WEB AUTHENTICATION PROTOCOL

The W3C Web Authentication protocol ensures *strong* user authentication using asymmetric cryptography [30]. W3C Web Authentication defines the security goals of their authentication protocol in terms of the protocol being resistant to *phishing attacks*, *man-in-the-middle attacks*, and *unlinkability* between user accounts and services.

A user is trying to authenticate to a *relying party* (RP) such as a website. The user is assumed to have one or more *authenticator*(s). For authenticating a user the authenticator has a set of *credential keys*, where new keys are created per origin, possibly derived via functions like key derivation (although this is unspecified) from a master key. Also, each authenticator has *attestation keys*, an asymmetric keypair for the identification of the authenticator or type of authenticator. A public key credential is created and stored by an authenticator for each relying party, where separate relying parties are defined by origins. An *origin* is defined as the domain name (including top-level domain), protocol, and port of a website such that https://mail.google.com has the origin *google.com* and thus shares a domain with https://maps.google.com but not with https://maps.apple.com or http://evil.com [26].

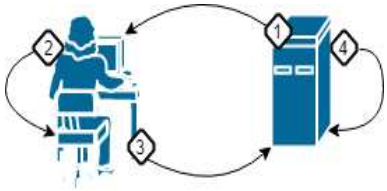

Figure 2: An example challenge-response flow

As given in Figure 2, authentication is done via cryptographically signing a challenge response protocol from the server. In detail, the private credential key never leaves the device but signs a challenge given by the server (1). A user interaction (such as a button press on a smartphone) is required for the user to sign the credential, which gives a "proof of presence" of a human being (2), as opposed to an automated attacker that is trying to brute force the signature from a captured device. Then the signed challenge is sent back to the server (3), registering the public key of the user or authenticating the user (4). The server challenge is a nonce, as well as its origin in order to prevent phishing and a HTTPS Token Binding (a number generated by the server that identifies a particular TLS channel) is given to prevent *man-in-the-middle attacks*. Unlinkability is preserved by having different credential keys for different origins. The public key credential can only be accessed by origins belonging to that Relying Party. These properties and their threat models are verified in Section 5.

The protocol has two distinct phases: 1) At least one registration phase to enroll a user's authenticator 2) One or more authentication phases, with a new one commencing when the user visits the website. During *registration*, a public key credential is created on a client's authenticator and associated by a relying party with the user's account on that relying party. In contrast, in the *authentication* phase, the challenge is sent from the relying party to the client and the relying party is presented by the client with an assertion proving the presence of the user who registered the public key credential and so authenticates the user. For signatures, the W3C Web

Authentication protocol uses ECDSA (Elliptic Curve Digital Signature Algorithm). Currently, W3C Web Authentication supports the NIST P-256 curve.

3.1 Registration

In the registration phase, the user and a Relying Party work in concert to create a public key credential and associate it with the RP's account for that user.

(1) The RP (server) sends a challenge (a nonce) and an RP identifier to the user's client. The browser checks to see the RP identifier matches the origin of the sender.
(2) The authenticator generates a new keypair for the origin of the RP, associating that credential keypair with that RP's identifier.
(3) The authenticator returns to the RP, via the browser, a signed attestation certificate that includes an attestation key for the authenticator in addition to the challenge and the user's public key associated with the RP, along with other associated data (i.e. the origin).
(4) The RP verifies the nonce and associated data before extracting the information in the attestation certificate. The RP associates the credential public key with the account (handle) of the user.

In the authentication phase, the authenticator attempts to cryptographically prove to a Relying Party that the user controls the credential private key associated with a previously-registered public key credential. A counter is added to make sure an authenticator is not "cloned" (when there are two different authenticators operating in parallel claiming to be the same authenticator).

(1) The RP sends a challenge (a nonce) and the RP identifier to the user along with a handle previously registered for a keypair by the RP. As in registration, the browser checks to see the RP identifier matches the origin.
(2) The browser prompts the user to select a credential, possibly from a list of credentials associated to the RP. The user consents for using this credential and generates an assertion signature with his private key for that credential.
(3) The authenticator signs the challenge and associated data.
(4) The authenticator, via the browser, returns a signed challenge and a counter to the RP.
(5) The RP verifies the challenge, the counter, and verifies the signature of the user using the credential public key previously associated during registration with their account.

4 PROVERIF

ProVerif [9] ("Protocol Verifier") is an automatic tool for formal verification under the symbolic Dolev-Yao [15]. In a Dolev-Yao model, the cryptographic primitives are assumed to be working and are called as functions that an attacker cannot change, although the attacker may use the results of these cryptographic functions. In the threat model of Dolev-Yao, the attacker can observe the channel between participants in a protocol and can read, alter, block, and inject messages.

$M, N ::=$	terms
x	variable
$a[M_1, \ldots, M_n]$	name
$f(M_1, \ldots, M_n)$	constructor
$F ::= pred(p_1, \ldots, p_n)$	fact
$R ::= F_1 \wedge \ldots \wedge F_n \Rightarrow F$	Horn clause

Figure 3: Syntax of Horn Clauses Representation

ProVerif takes as a representation of a protocol described by Horn clauses and automatically tries to prove the security properties of the protocol by simulating an attacker over multiple sessions of the protocol. ProVerif can create generic predicates of any arity, but ProVerif includes special built-in predicates for cryptography defined with equational theories, including symmetric public-key cryptography (encryption and signatures), hash functions, and Diffie-Hellman key agreement. It can handle an unbounded number of sessions of the protocol and an unbounded message space. ProVerif can prove the following properties:

- **secrecy**: The attacker cannot obtain a secret.
- **authentication**: Modelled as a a correspondence property, namely that one event implies another event. In the case of authentication, a server accepts authentication if and only if the user accepts the same authentication. Informally, the user cannot authenticate to another server, or an attacker cannot impersonate a user.
- **strong secrecy**: The attacker can not identify the difference between any session of the protocol if the value of the secret changes.
- **equivalences**: Processes that can be proven to differ only by terms.

Due to its formalism, ProVerif can generate false attacks but if ProVerif claims that the protocol satisfies some property, then the property is actually satisfied. When the ProVerif cannot prove a property, it tries to reconstruct an attack by automatically generating an execution trace of the protocol that falsifies the desired property.

4.1 Horn clauses

A protocol in ProVerif is represented by a set of Horn clauses; the syntax of these clauses is shown in Figure 3 and they have a standard Horn Clause formal semantics [9]. In Figure 3, x ranges over variables, a over constants (called "names"), f over functions, and p over predicates. The term M represent messages that are exchanged between participants of the protocol. A variable can represent any name. Names represent atomic values, such as keys and nonces. Each participant has the ability to create new names: fresh names are created at each run of the protocol. Names are considered as functions of the messages previously received by the principal that created the name. Thus, names are distinguished only when preceding messages are different in a protocol session. The *security properties* are given as queries over the Horn clauses, i.e. Horn clauses that may be true or false.

$P, Q, R ::=$	processes
0	null process
$P\|Q$	parallel composition
$!P$	replication
$if\ M = N\ then\ P\ else\ Q$	conditional
$\nu n.P$	name restriction

Figure 4: Syntax of the applied π calculus

4.2 The applied π calculus

As ProVerif describes multiple phases of a protocol with possible branches, it describes these phases (i.e. the execution of a protocol described via Horn clauses over multiple sessions) as processes using the applied π calculus [8] introduced by Abadi and Fournet [1]. The applied π calculus is an extension of the π calculus with function symbols defined by an equational theory. Abadi and Fournet added cryptographic functions using equational theories [1], whose equational theories are used to define cryptographic functions in ProVerif. Note that when ProVerif is run, these statements in the π calculus are transformed into Horn clauses.

In particular, we will use the *senc* (symmetric encryption) and *sdec* (symmetric decryption) cryptographic functions to model TLS and *sign* and *checksign* to sign and check public key signatures. We will also use the built-in type *nonce* for nonces. Under a Dolev-Yao model, a *bitstring* type is used for generic messages, such as a credential. The protocol as a process initially runs in the first phase; then at some point protocol in this phase can stop and processes of the next phase are run and so on. State may be passed between phases, such as the transfer of compromised keys.

To put it all together, a ProVerif protocol description Σ has two major parts:

- A sequence of Horn clauses $\Delta_1 \ldots \Delta_n$ that contain names (variable and constant), constructors and destructors, equations (needed by cryptographic functions), processes and queries. Queries define the security and privacy properties to prove.
- P, the top-level process which then uses $\Delta_1 \ldots \Delta_n$ as its toolkit for creating a flow for the protocol.

The replication $!P$, representing an unbounded number of copies of P in parallel, launches the protocol flow. During verification, tables store persistent state such that the process insert $a(M_1, \ldots, M_n); P$ inserts the entry $(M_1, \ldots, M_n)$ in Table a, and runs P. The process get $a(=M_1, x_2, \ldots, x_n)$ in P retrieves an entry $(N_1, \ldots, N_n)$ in Table a when $N_1 = M_1$. If such an entry is found, then $x_2, \ldots, x_n$ is bound to $N_2, \ldots, N_n$ and runs P. Tables can be used to store values such as nonces and secrets that maintain state between multiple runs of the protocol.

When ProVerif is used to formally verify the security properties of a protocol, it runs a *resolution algorithm* on the representation of the protocol, given as Horn clauses ($\Delta_1 \ldots \Delta_n$). A query of a security property is then Δ_q, i.e. a statement of a successful attack. Usually, this is *attacker(K)*, where the statement is that the attacker knows secret material K. The resolution algorithm tries to derive Δ_q from $\Delta_1 \ldots \Delta_n$. If it does, then the execution trace of the derivation is the *attack*. If it does not find an attack, the query is false. Note that ProVerif is *sound* such that if an attack exists in the protocol, it is found by the resolution algorithm. However, ProVerif is *incomplete* such that false attacks can be found (i.e. each attack should be manually checked) and insofar as for certain protocols ProVerif may not terminate.

5 FORMAL VERIFICATION OF W3C WEB AUTHENTICATION

In this section, we describe the process of formally verifying the W3C Web Authentication using ProVerif. For each section, we describe the threat model and then demonstrate the protocol's formalization using ProVerif, and finally if the properties were proven to hold by process of formal verification. In detail, we describe how we model the user and the server for both authentication and registration phases, and how we formalize privacy in terms of the same origin restriction in Section 5.3. The description of the protocol in ProVerif is given in boxes.

5.1 Components

In more detail, the following key material-related components are used in W3C Web Authentication. Given the complexity of W3C Web Authentication, we do not verify many components that are not used in key material-based operations. Note that as ProVerif does not have a representation of numbers so we cannot verify the counter, so it is left out of the model. We assume that the channel between the token and the browser is secured and so model both as a singler user entity. We assume the network connection is secured successfully using TLS, which we explicitly model via symmetric key encryption. The following variables in our model are used in both phases:

- **Credential public key** (pk_A): "The public key portion of a Relying Party-specific credential key pair, generated by an authenticator and returned to a Relying Party at registration time" [30]. For each relying party, the user A generates a new credential public key for future use.
- **Credential private key** (sk_A): "The private key portion of the credential key pair is known as the credential private key" [30]. The authenticator sign each message to a RP with this key, and it's usually saved in the authenticator or in the user A's platform.
- **Credential ID** ($credential_{ID}$): "A probabilistically-unique byte sequence identifying a public key credential source and its authentication assertions" [30]. This uniquely identifies a credential.
- **Relying Party Identifier** (RP_{id}): "A valid domain string that identifies the Relying Party" [30], with the domain by default being the origin of the RP (server S).
- **Nonce** (N_S): A nonce is an arbitrary number that may only be used once to prevent replay attacks, and used as part of the challenge in the WebAuth protocol sent by the server S.
- **Attestation certificate** ($sign(att)$): "An attestation statement contains a signature by an attestation private key over the attested credential public key and a challenge, as well as a certificate or similar data providing provenance information for the attestation public key, enabling the Relying

Party to make a decision to trust the attestation or not"' [30]. We simplify the model of the specification by considering the attestation object in the registration as a single signed object including the attested credential data and the challenge.

- **Attestation public key** ($attpk_A$): "The attestation public key conveyed in the attestation certificate to verify the attestation signature" [30], of user A for attestation att.
- **Attestation private key** ($attsk_A$): The private key needed to "sign the Relying Party-specific credential public key (and additional data) that it generate" [30] of user A for attestation att.

5.2 Security Properties of W3C Web Authentication

In this section, the ProVerif model is described. Code samples in ProVerif syntax are given in boxes, illustrating the parts of the model as needed.[6]

5.2.1 Threat Model. The threat model of security properties is a *network attacker*, an attacker that can control all messages on a channel [2]. In *passive* attacks, the attacker can intercept and read all messages sent on the network and can run arbitrary computations on messages. In *active* attacks, the attacker can compute messages and also alter, block, and send its own messages on the network, including messages involving computation.

5.2.2 Security Goals. We want to prove integrity, which we can prove by showing that no secret key material can be known by the attacker, as is done via "phishing attacks," and so all messages maintain integrity. We also want to prove authentication, i.e. the user can only authenticate to the server they have registered to (preventing "man in the middle" attackers) so the attacker can not impersonate the user.

5.2.3 ProVerif Description. In our model, we assume that a valid TLS connection is established before commencing the W3C Web Authentication protocol. As the network traffic is encrypted by TLS, we model that each message sent on the channel is encrypted via a symmetric shared secret. We do not model HTTPS Token Binding and other associated non-cryptographic data outside the RP ID. To simplify the explanation, we use the term "server" rather than "relying party" (RP) in our model and do not model provenance data (outside of the RP ID) like Token Binding.

During the registration phase the server (RP) S sends a challenge and RP_{id} to the user A. The user generates the attestation certificate, signs it with their attestation secret key, and sends back to the server.

$$\begin{aligned} S &\rightarrow A : (N_s, RP_{id}) \\ A &\rightarrow S : sign(N_s, RP_{id}, pk_A \\ &\qquad credentialID, attpk_A), attsk_A) \end{aligned}$$

When the server receives the attestation certificate, the server verifies the certificate of the authenticator (via the signature) and the challenge as well as RP ID.

[6] The full ProVerif code is available from https://github.com/hhalpin/webauthn-model.

In the authentication phase, the server sends another challenge to the user. The user receives another challenge, signs it and returns a response signed with the credential public key to the server (checking origin, although we do not model any other additional data such as HTTPS Token Binding and user handles associated with Credential IDs).

$$\begin{aligned} S &\rightarrow A : (N_s, RP_{id}) \\ A &\rightarrow S : sign((N_s, RP_{id}), sk_A) \end{aligned}$$

The server verifies the signature with the user's public key for his credential. Once the signature is verified, the user is authenticated.

In our ProVerif model, subscripts are concatenated to names in variables (i.e. $N_s \rightarrow Ns$). The symmetric key established by TLS is given by k.

To model W3C Web Authentication in ProVerif, we query whether the attacker can get knowledge of any key material or spoof the Credential ID via the following queries: *query attacker(attskA)*, query *attacker(attpkA)*, *query attacker(k)*, and *query attacker(credentialID)*. To verify the authentication property, we created an authentication procedure with two events, *sentChallengeResponse* and *validChallengeResponse*. The first event triggers the second event if the authentication is successful in order to prove authentication.

```
event sentChallengeResponse(bitstring, bitstring).
event validChallengeResponse(bitstring, bitstring).
query m1:bitstring, m2:bitstring;
event(validChallengeResponse(N, s)) ⟹
    event(sentChallengeResponse(m1, m2)).
```

A user is modelled as a set of credential keys and attestation keys on their authenticator. Credential keys are given a credential ID.

```
let processUser (
k: key, credentialID:bitstring,
attskA: AttestationPrivatekey,
    attpkA:AttestationPublicKey) =
```

A user has two functions, one for registration and another for authentication. If a user receives a challenge nonce, it responds with the signed attestation credential if they have not registered with the server before. Therefore, the user generates a new credential keypair (pk_A, sk_A) and returns an attestation credential signed with their attestation secret key $attsk_A$.

```
in(c, s:bitstring);
let(Nu:nonce, RP:host) = sdec(s, k) in
new skA:skey;
let pkA = spk(skA) in
out(c, senc(signAtt((spk(skA),
attpkA, credentialID, Nu), attskA), k));
```

If a user receives a challenge nonce and it has received the message before, then it returns the attestation credential signed with

its already created credential key sk_A.

```
in(c, s:bitstring);
let mess1 = sdec(s, k) in
event sentChallengeResponse(mess1,
sign(senc(mess1, k), skA));
out(c, sign(senc(mess1, k), skA)).
```

The second party in the Web Authentication protocol is the server, which is identified with its "origin" RP_{id} and has a shared symmetric TLS key k with the user.

```
let processServer (k: key,
RP_id:host) =
```

During registration, the server sends a nonce N_{s1} with its RP_{id}. When it receives a message, it stores the received credential key pk_{Y1} to associate it with user A and checks to make sure the attestation certificate is signed by the attestation key $attpk_{u1}$.

```
new Ns1 : nonce;
out(c, senc((Ns1, RP_id), k));
in(c, s:bitstring);
let m = sdec(s, k) in
let (pkY1: pkey, attpkU1:AttestationPublicKey,
credUser: bitstring, Nu:nonce) = getmessAtt(m) in
let ver = checksignAtt(m, attpkU1) in
```

During authentication, the server checks that the received nonce N_u matches N_{s1}. Then the server checks to make sure the signed challenge returned by the user is signed by the key pk_{Y1}. Only then is the authentication event *validChallengeResponse* triggered.

```
if (Nu = Ns1) then
(new Ns2 : nonce;
let mess = nonce_to_bitstring ( Ns2 ) in
out(c, senc ((mess, RP_id), k));
in(c, s:bitstring);
let m1 = sdec(s, k) in
let m2 = checksign(m1, pkY1) in
let m3 = getmess(s) in
if (m3 = getmess(s))
then    event validChallengeResponse(m3, s))
```

In order to determine if an attacker can impersonate a user via a "phishing" or "man in the middle attack," we can query to determine if an attacker can reach the authentication event, as could be done if the attacker could gain knowledge of the key material of the user and so register themselves as the user (phishing), or intercept the challenge and send their own signature which would be accepted by the server as the user's signature ("man in the middle"). The results of running ProVerif over the following model resulted in no knowledge of the key material or credential ID being found by the attacker, and a *validChallengeResponse* from the server always was preceded by a *sentChallengeResponse* from the user. Therefore, no phishing and "man in the middle" attacks were found, so we can surmise that the W3C Web Authentication securely authenticates users to servers.

5.3 Privacy Properties of Web Authentication

While W3C Web Authentication has been formally verified to uphold its security properties of preventing phishing and "man in the middle" attacks, W3C Web Authentication also claims that authentication is unlinkable between origins, and we now seek to formally verify this property.

5.3.1 Threat Model. Traditionally cryptographic protocols have been required to satisfy secrecy (preventing phishing) and authentication (preventing "man in the middle") properties. These requirements are normally verified by modeling them as *reachability problems*, i.e. can an attacker reach the key material or authenticate as a user? Some properties such as privacy cannot easily be expressed using the traditional threat model of ProVerif. However, similar to key material in security, once sensitive information is leaked, the identity of a user may be revealed. The threat model for the privacy properties is a malicious or honest but curious server whose goal is to uniquely identify a user, which is not a network attacker, but a *Web attacker* [2]. The goal of the Web attacker is to identify the user across domains using techniques like cookies to track the user. The goal of new web standards is to avoid accidentally introducing a new "super-cookie" by accident that lets even non-malicious servers track users across origins.

5.3.2 Security Goals. Privacy is difficult to formalize insofar as privacy is a holistic concept and may mean different things in different scenarios. Typically, privacy is defined as *unlinkability*, where given "two or more items of interest" that "these items of interest are no more and no less related after [the attacker's] observation than they are related concerning his a priori knowledge" [24]. In terms of the same origin policy, this means that for any authentication event using Web Authentication within a given origin, the user should not be related in anyway to another event using Web Authentication at another origin. In terms of unlinkability, a user being unlinkable in relationship to other users creates an *anonymity set* as regards a group of users and a given action like authentication. As put by Pfitzmann, given that "anonymity is the state of being not identifiable within a set of subjects, the anonymity set," various technologies to maintain privacy may be considered better than others insofar as they provide larger anonymity sets. However, for purposes of our formal verification, we will not use anonymity sets, but simply see if a user can be linked at all between two authentication events.

5.3.3 ProVerif Description. Unlike security properties, there is not a clear way to model privacy properties in ProVerif. One approach is to compare an ideal flow of the protocol from one that instantiates concrete values, so that privacy violations can be inferred if a difference between the ideal and the instantiated flow can be detected [20]. However, the problem with this approach is that it assumes that the attacker is a network attacker observing

the secure channel (such as the TLS connection), not a party in the protocol such as the server. One simple option of modeling the server itself as malicious could have the server drop all data to the clear, but this would not accurately model a protocol that inadvertently allows some data to be shared between non-malicious servers on different origins due to the design of the protocol itself.

In order to model privacy in terms of the protocol itself violating the same origin property in ProVerif, we used a reachability event. For a two-party protocol, a unique identification of a user simply requires that between two sessions of the same program the same user can be identified via reaching an event that connects two different registration or authentication events in the protocol. In order to test unlinkability in Web Authentication, we add to the server the ability to create a database of keys using ProVerif's *table* constructor. For each registration phase, the servers checks its database of attestation keys and tries to determine if the same attestation key has been used before. If it has, the user is *re-identified* by virtue of using the same attestation key and the unlinkability property is broken. For example, this is the case when the same user tries to create two different accounts with different credential keys but using the same authenticator (if the authenticator has a single attestation key).

In ProVerif, we model privacy as the event *reachSameKey*, where we store key materials such as attestation keys ($Attpk_U$) for a given user U in a table. Then we can show that when the model is ran over an unbounded number of session, the same key material can be reached across multiple sessions, allowing the attestation key material to be used as an identifier for a user across origins.

```
get tableAttPk(attpkU) in
  event reachSamekey(attpkU)
else
 insert tableAttPk(attpkU);
```

ProVerif proved that *reachSameKey* can be reached across multiple sessions of the protocol.[7] The problem is that the W3C Web Authentication protocol sends in the registration phase an attestation public key and the initial attestation credential is signed by the attestation private key. The server simply has to store these attestation public keys and check to see if an attestation key already exists in its database when a user attempts to register to the server. If the attestation key is already in the database, the user is *reidentified* by their re-use of the same authenticator. Note that keys are not required by this attack on privacy, as any authenticator-specific identifier would allow the attack to be done.

6 RESULTS AND NEXT STEPS

We modeled both the security and privacy properties of W3C Web Authentication, and proven (assuming TLS usage) that the user can authenticate securely using cryptographic key material as their key material can not be violated by a network-level attacker. However, if we assume that the server is either "honest but curious" or malicious, the privacy property of unlinkability between origins can be violated. In this second scenario, the server is trying to identify that the same user is behind two accounts even though he is using two different public credential keys. The server creates a database and saves the attestation public key during the registration phase. When the same user is registering their authenticator for the second time and tries to create an account with another credential public key, the server checks the database and identifies that it's the same user as the new user registered the same authenticator as a previous user. This scenario can be expanded if two servers are co-operating. In this case, the first server (on one origin) can share the attestation key of the device with a second server controlling a different origin than the first server. Then, when a user that has already registered their authenticator at the first server begins to register their authenticator on to the second server, the second server can identify the user across origins. Therefore, the privacy properties of W3C Web Authentication do not hold unless there is a modification to the authenticator's attestation keys cannot be linked across origins with a user identity. The specification does note that no attestation may be used and that basic attestation should have keys where "authenticators of the same model often share the same attestation key pair" [30], but this is not enforced. The idea of an "Anonymization CA" that "can dynamically generate per-origin attestation keys and attestation certificates" is mentioned but "the envisioned functionality of an Anonymization CA is not firmly established" [30]. There are three options that W3C Web Authentication currently discusses, although these methods are not given as mandatory in the W3C draft Recommendation. In detail:

- In *self-attestation*, the authenticator "uses the credential private key to create the attestation signature"and so "the authenticator has no distinct attestation key pair nor attestation certificate" [30], allowing the attestation not to convey any new property about the authenticator and so would preserve the user's unlinkability across origins.
- Even under basic authentication, it is possible that "authenticators of the same model often share the same attestation key pair" [30]. Therefore, rather than uniquely identifying a user, the use of the attestation key likely will reduce their anonymity set.
- The authenticator has an "authenticator-specific (endorsement) key" that is "used to securely communicate with a trusted third party, the Attestation CA" [30]. However, this would reduce the anonymity set of the user and also has the unfortunate side-effect of introducing a trusted third-party into the system that can then vouch for any details of the authenticator. The main advantage of the CA is that rather than per model keys, possibly per brand keys could be used that would increase the anonymity set, i.e. all Yubikeys could use Yubico as their trusted Privacy CA, producing a larger anonymity set.
- With *elliptic curve direct anonymous attestation* (ECDAA), an anonymous authentication credential is used to blindly sign the attestation credentials. The use of "blinding avoids the DAA credentials being misused as global correlation handle" [30] and so keeps the user's attestation action private from the CA itself.

[7] The full ProVerif output for the attacks on both models are available from https://github.com/hhalpin/webauthn-model.

ECDAA is a variant of Direct Anonymous Attestation (DAA) that uses elliptic curves rather than the RSA keys typically used in DAA [11]. Of all the schemes suggested by the W3C Web Authentication Protocol [30], the most promising in terms of privacy is ECDAA, so we will explore ECDAA as it is the best mitigation available currently for the privacy problem of W3C Web Authentication. The DAA [14] protocol satisfies privacy whenever a process where Alice interacts with the verifier is observationally equivalent (to an attacker whose goal is identify users) to when Bob interacts with the verifier. This is the only technique that guarantees the identity of the user are not revealed to a third party (such as in the attestation CA case) and the attestation of the authenticator can be transmitted to a server without linking it across origins or to the third party issuer.

6.1 ECDAA

DAA (and ECDAA) allows the remote attestation of an authenticator to a server while preserving the privacy of user, unlike the normal case of an attestation CA. The goal is that a user can have the capabilities of their authenticator attested while protecting their identity from a third-party server, called the *issuer*, that generates the attestation credentials. The server must only learn that their authenticator is trusted and not which particular authenticator is being used, and therefore it would preserve the same origin policy restriction of W3C Web Authentication. The DAA protocol can be viewed as a group signature scheme without the ability to revoke anonymity. In broad terms, the authenticator contacts the issuer of attestation credentials (such as their device manufacturer) for their device and requests membership to a group. If the issuer wishes to accept the request, it grants the authenticator an attestation credential. The host is now able to anonymously authenticate itself as a group member to a server, and the server can act as a verifier with respect to its attestation credential. This protocol is initiated when an authenticator wishes to obtain an anonymous authentication credential and so must take place before the registration and authentication phases of W3C Web Authentication.

(1) The authenticator creates a secret f and a blinding factor v' and it constructs a blinding message $U = blind(f, v')$.
(2) The issuer creates a nonce n_e.
(3) The authenticator receives the nonce in an encrypted message, decrypts it and returns $hash(U||n_e)$.
(4) The issuer confirms that the hash is correctly formed.
(5) The issuer generate a nonce n_i and sends to the authenticator.
(6) The authenticator constructs a proof of knowledge that the messages U and n_i are correct.
(7) The issuer verifies the proof and generates a blind signature on the message U
(8) The authenticator verifies the issuer's signature and proof and unblinds the signature, revealing a secret credential v (the signed f).

Once the authenticator has the anonymous attestation credential from the issuer, it can produce a proof of knowledge of a signature on the message sent by the server and therefore proves that the authenticator has a valid attestation from the issuer. This technique has been formally verified in the DAA case using ProVerif [14], although the formalization of the elliptic curve-variant remains future work.

Note that signature of the issuer reduces the anonymity set, as the issuer reveals "per brand" identity, and from the standpoint of the RP, the anonymity set could be the same as an attestation CA. What is likely needed for future versions of the W3C Web Authentication API, in particular on anonymization CAs, is a move to finer granularity of attributes, such as the support of simply revealing the particular attributes of authenticators. Regardless of the scheme in W3C Web Authentication used to mitigate the attack, the attestation public key and attestation certificate on registration in W3C Web Authentication still reduces the anonymity set considerably and increases the chances of linking a user across origins. Therefore, future research should engage with attribute-based credentials and forms of unlinkable credentials that have better privacy properties than DAA-based schemes [17].

7 CONCLUSION

While we have started to formally verify that the W3C Web Authentication protocol is secure and so ensures strong user authentication, the W3C Web Authentication API allows violations of the privacy of users that undermine the stated goals of the protocol in terms of unlinkability of users between origins, unless suggested functionality is made to be mandatory. The solution is to require more detailed and mandatory privacy protections in the protocol itself, such as having the authenticator generate dynamically an attestation public key that is specific to each origin where an identifier guarantees the type of authenticator (rather than a separate key) or use cryptographic techniques such as ECDAA or zero-knowledge proofs that are more privacy-preserving than any of the options recommended in the W3C Web Authentication protocol [17]. Future work should formalize more completely ECDAA with W3C Web Authentication and the other options in the specification. In general, our model makes a number of simplifications and so much work is needed to formally model the complete protocol as given in the W3C specification, and so this model should be considered a beginning that can be subject to review by the W3C Working Group rather than the end of analysis.

Although security properties are fairly straightforward to prove, privacy properties are more difficult to formalize, and future research should find better formalizations that fit within both symbolic and computational formal verification tools, with a focus on common privacy boundaries such as the same origin policy for the Web. This would allow multiple protocols from web browsers that should maintain the same origin policy, such as the currently unverified WebRTC protocol and verified (although not in terms of privacy properties) W3C Web Cryptography API [10], to be easily verified using the same tool-set. Likely a modular approach to formally verifying the security and privacy properties of web applications could eventually be accomplished, as these web applications depend on combinations of these APIs and protocols. More work needs to be done to model the same origin policy in ProVerif even for W3C Web Authentication, in order to develop a standard way of verifying privacy across Web APIs. More comprehensive ways of modeling privacy beyond reachability should be applied and extended to cover the same origin policy [20], although we are the

first to our knowledge to model privacy via the same origin policy as distinct from network attackers in a formally verified Web API.

In the future, all protocols should go through a rigorous process of formal verification in order to determine their security and privacy properties before standardization. This would have a number of effects that are beneficial to protocol design. Tools such as ProVerif are specialized in security and privacy properties, and so can be deployed at the very beginning of the design phase of new protocols before or at the beginning of standardization. The widespread use of formal verification would not only increase security and privacy, but would allow the process of standardization to be less dependent on human reputation or the influence of powerful implementers, as has been claimed to be the case with controversial standards around DRM like the W3C Encrypted Media Extensions standard [19]. The process of the standardization should be moved from being a social process to a scientific process, where the security and privacy of standard protocols can be proven via automated formal verification, with proofs subject to public review. The scientific analysis of security and privacy in standards via formal verification presents an exciting frontier in the science of security.

8 ACKNOWLEDGMENTS

Dirk Balfanz (Google) provided the image in Figure 2. The authors are funded in part by the European Commission H2020 NEXTLEAP Project (Grant No. 6882).

REFERENCES

[1] Martín Abadi and Cédric Fournet. 2001. Mobile values, new names, and secure communication. In *ACM SIGPLAN Notices*, Vol. 36. ACM, 104–115.

[2] Devdatta Akhawe, Adam Barth, Peifung E Lam, John Mitchell, and Dawn Song. 2010. Towards a formal foundation of web security. In *Computer Security Foundations Symposium (CSF), 2010 23rd IEEE*. IEEE, 290–304.

[3] José Bacelar Almeida, Manuel Barbosa, Gilles Barthe, and François Dupressoir. 2016. Verifiable Side-Channel Security of Cryptographic Implementations: Constant-Time MEE-CBC. In *Fast Software Encryption - 23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers (Lecture Notes in Computer Science)*, Thomas Peyrin (Ed.), Vol. 9783. Springer, 163–184.

[4] Dirk Balfanz, Adam Langley, Niel Harper, and Jeff Hodges. 2017. *Token Binding over HTTP.* IETF. https://datatracker.ietf.org/doc/draft-ietf-tokbind-https.

[5] K Bhargavan, B Blanchet, and N Kobeissi. 2016. Verified Models and Reference Implementations for the TLS 1.3 Standard Candidate. In *IEEE Symposium on Security and Privacy (Oakland)*.

[6] Karthikeyan Bhargavan, Antoine Delignat Lavaud, Cédric Fournet, Alfredo Pironti, and Pierre Yves Strub. 2014. Triple handshakes and cookie cutters: Breaking and fixing authentication over TLS. In *Security and Privacy (SP), 2014 IEEE Symposium on*. IEEE, 98–113.

[7] Bruno Blanchet. 2007. CryptoVerif: Computationally sound mechanized prover for cryptographic protocols. In *Dagstuhl seminar "Formal Protocol Verification Applied*. 117.

[8] Bruno Blanchet et al. 2016. Modeling and verifying security protocols with the applied pi calculus and ProVerif. *Foundations and Trends® in Privacy and Security* 1, 1-2 (2016), 1–135.

[9] Bruno Blanchet, Ben Smyth, and Vincent Cheval. 2015. Proverif 1.90: Automatic cryptographic protocol verifier, user manual and tutorial. *URL: http://prosecco. gforge. inria. fr/personal/bblanche/proverif/manual. pdf*.

[10] Kelsey Cairns, Harry Halpin, and Graham Steel. 2016. Security Analysis of the W3C Web Cryptography API. In *Security Standardisation Research*. Springer, 112–140.

[11] Liqun Chen, Dan Page, and Nigel P Smart. 2010. On the design and implementation of an efficient DAA scheme. In *International Conference on Smart Card Research and Advanced Applications*. Springer, 223–237.

[12] Alexei Czeskis, Michael Dietz, Tadayoshi Kohno, Dan Wallach, and Dirk Balfanz. 2012. Strengthening user authentication through opportunistic cryptographic identity assertions. In *Proceedings of the 2012 ACM conference on Computer and communications security*. ACM, 404–414.

[13] Anupam Das, Joseph Bonneau, Matthew Caesar, Nikita Borisov, and XiaoFeng Wang. 2014. The Tangled Web of Password Reuse.. In *NDSS*, Vol. 14. 23–26.

[14] Stéphanie Delaune, Mark Ryan, and Ben Smyth. 2008. Automatic verification of privacy properties in the applied pi calculus. *Trust Management II* (2008), 263–278.

[15] D. Dolev and A. Yao. 1983. On the security of public key protocols. *Information Theory, IEEE Transactions on* 29, 2 (March 1983), 198 – 208. https://doi.org/10.1109/TIT.1983.1056650

[16] Shafi Goldwasser and Silvio Micali. 1984. Probabilistic encryption. *Journal of computer and system sciences* 28, 2 (1984), 270–299.

[17] Jan Hajny and Lukas Malina. 2012. Unlinkable attribute-based credentials with practical revocation on smart-cards. In *International Conference on Smart Card Research and Advanced Applications*. Springer, 62–76.

[18] Harry Halpin. 2012. Web Authentication: The next step in the evolving identity eco-system?. In *Proceedings of the IEEE CS Security and Privacy Workshops*.

[19] Harry Halpin. 2017. The Crisis of Standardizing DRM: The Case of W3C Encrypted Media Extensions. In *International Conference on Security, Privacy, and Applied Cryptography Engineering*. Springer, 10–29.

[20] Lucca Hirschi, David Baelde, and Stéphanie Delaune. 2016. A method for verifying privacy-type properties: the unbounded case. In *Security and Privacy (SP), 2016 IEEE Symposium on*. IEEE, 564–581.

[21] Gavin Lowe. 1996. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *International Workshop on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 147–166.

[22] Roger M Needham and Michael D Schroeder. 1978. Using encryption for authentication in large networks of computers. *Commun. ACM* 21, 12 (1978), 993–999.

[23] Olivier Pereira, Florentin Rochet, and Cyrille Wiedling. 2017. Formal Analysis of the FIDO 1. x Protocol. In *International Symposium on Foundations and Practice of Security*. Springer, 68–82.

[24] Andreas Pfitzmann and Marit Hansen. 2005. Anonymity, unlinkability, unobservability, pseudonymity, and identity management-a consolidated proposal for terminology. (2005).

[25] Jonathan Protzenko, Jean-Karim Zinzindohoué, Aseem Rastogi, Tahina Ramananandro, Peng Wang, Santiago Zanella-Béguelin, Antoine Delignat-Lavaud, Cătălin Hriţcu, Karthikeyan Bhargavan, Cédric Fournet, et al. 2017. Verified low-level programming embedded in F. *Proceedings of the ACM on Programming Languages* 1, ICFP (2017), 17.

[26] Sakthivel Ramachandran Vadivel Saravanan Subramanian. 2006. Web Security: Same Origin Policy and its Exemptions. (2006).

[27] Sampath Srinivas, Dirk Balfanz, and Eric Tiffany. 2015. *Universal 2nd Factor (U2F) overview*. FIDO Alliance Proposed Standard.

[28] Nikhil Swamy, Juan Chen, Cédric Fournet, Pierre-Yves Strub, Karthikeyan Bhargavan, and Jean Yang. 2011. Secure distributed programming with value-dependent types. In *Proceeding of the 16th ACM SIGPLAN international conference on Functional Programming*. 266–278.

[29] Bharadwaj. Vijay, Hubert Le Van Gong, Dirk Balfanz, Alexei Czeskis, Arnar Birgisson, and Jeff Hodges. 2015. FIDO 2.0: Web API for accessing FIDO 2.0 credentials. (2015). https://www.w3.org/Submission/2015/SUBM-fido-web-api-20151120/.

[30] Bharadwaj. Vijay, Hubert Le Van Gong, Dirk Balfanz, Alexei Czeskis, Arnar Birgisson, Jeff Hodges, Michael Jones, Rolf Lindeman, and J.C. Jones. 2017. Web Authentication: An API for accessing Public Key Credentials Level 1. (2017). https://www.w3.org/TR/webauthn/.