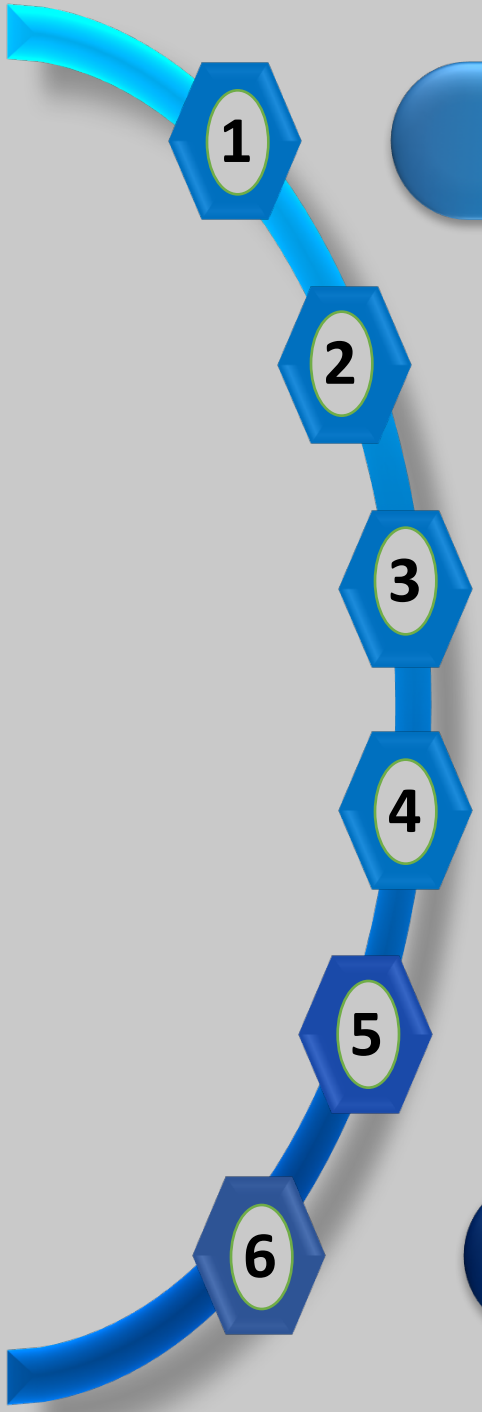
Formal Verification of the W3C Web Authentication Protocol

Authors:

Harry Halpin
INRIA, Paris
Harry.halpin@inria.fr

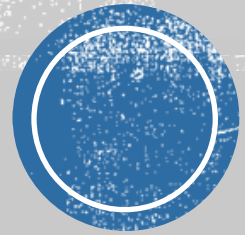
Iness Ben Guirat
INSAT, Tunis
inessbenguirat@gmail.com

Plan



1. Background
2. Web Authentication Protocol
3. ProVerif
4. Formal Verification
5. Results and next steps
6. Conclusion

Background



→ **Web Authentication**

?

→ PhoneAuth

→ FIDO specifications

Web Authentication Protocol

The W3C Web Authentication protocol ensures strong user authentication using asymmetric cryptography

- ✓ Phishing attacks
- ✓ Man-in-the-middle attacks
- ✓ Unlinkability

When a user is trying to authenticate to a RP

- A user has one or more authenticator(s)
- Each authenticator has *attestation keys*
- The authenticator has a set of *credential keys*

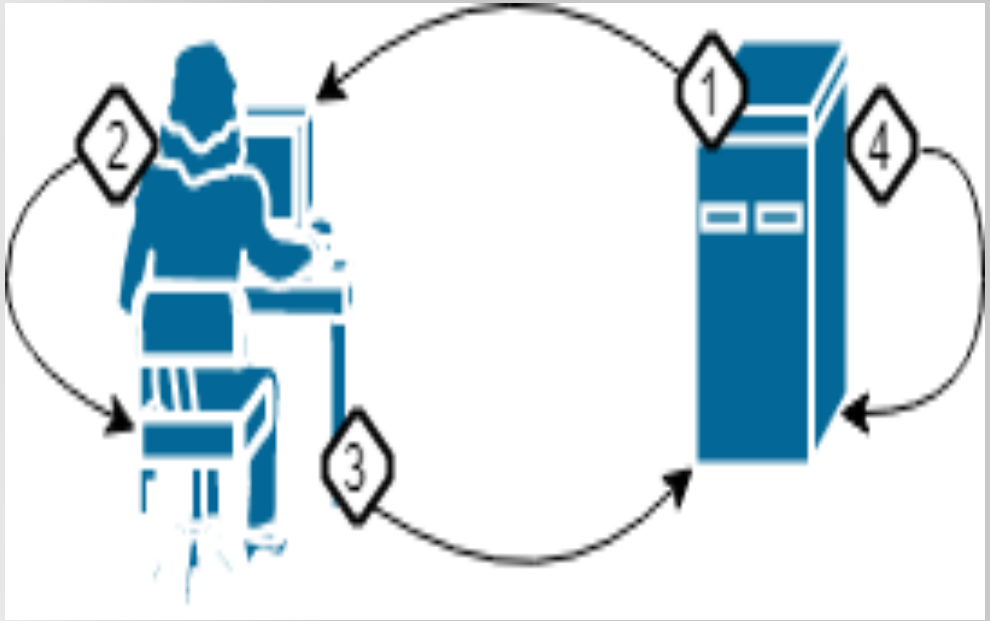
Authentication

1. The RP sends a challenge 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 key pair for the origin of the RP, associating that credential key pair with that RP's identifier.
3. The authenticator returns to the RP 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 of the user

Registration

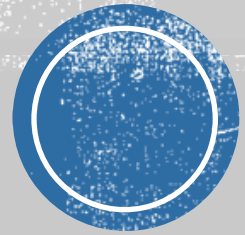
1. The RP sends a challenge (a nonce) and the RP identifier to the user along with a handle previously registered for a key pair by the RP. As in registration, the browser checks to see the RP identifier matches the origin.
2. The browser prompts the user to select a credential, possibly from a list 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.

A Challenge Response Protocol



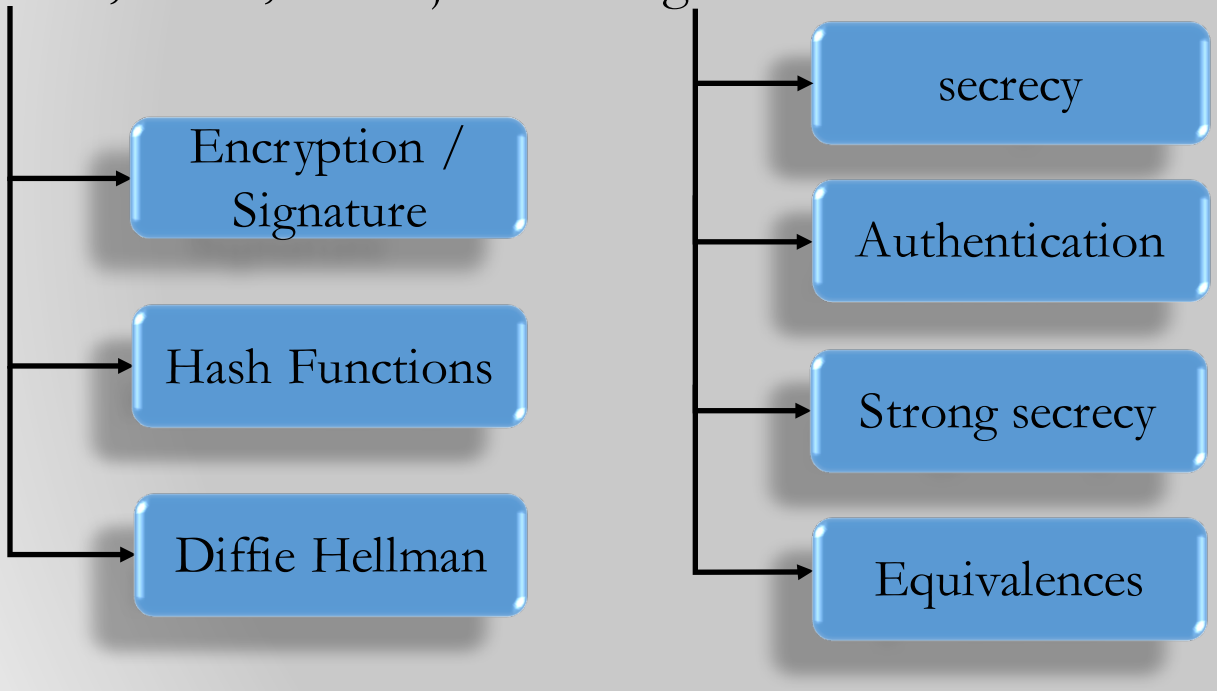
1. The private credential key(that never leaves the device) signs a challenge given by the server.
2. A user interaction (such as a button press on a smartphone) is required for the user to sign the credential, which gives a "proof of presence" of a human being.
3. The signed challenge is sent back to the server.
4. Registration/Authentication.

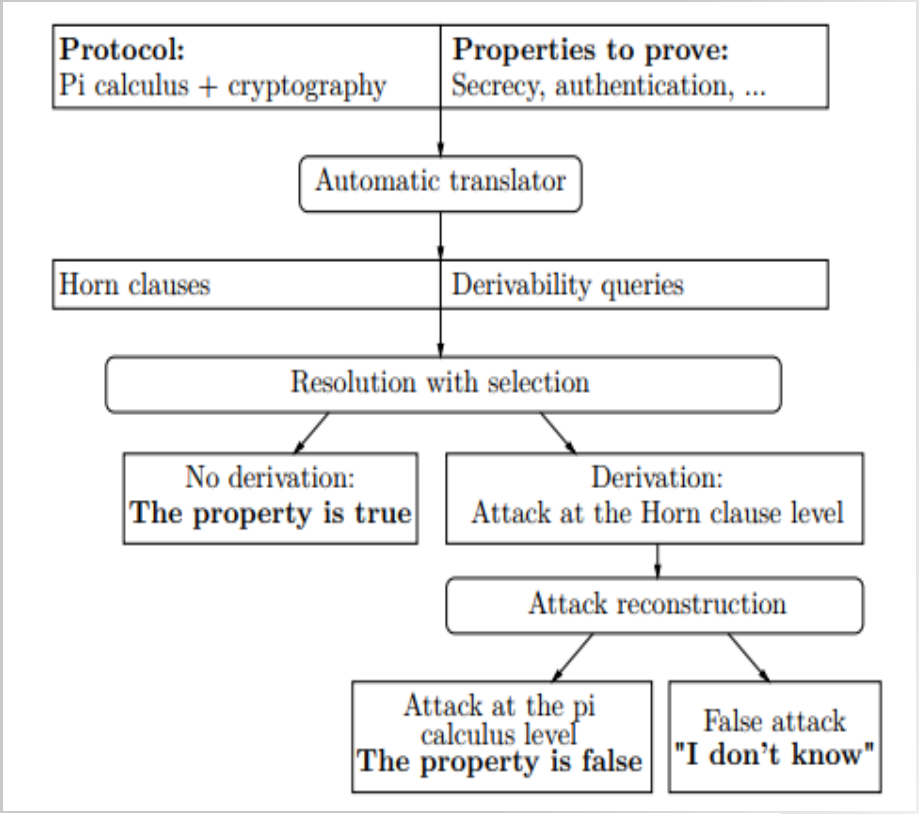
ProVerif



ProVerif

ProVerif ("Protocol Verifier") is an automatic tool for formal verification under the symbolic Dolev-Yao.
In a Dolev-Yao model, the cryptographic primitives are assumed to be working 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 the participants and can read, alter, block, and inject messages.





Horn Clauses

A protocol in ProVerif is represented by a set of Horn clauses.

$M, N ::=$	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

The applied π-calculus

As ProVerif describes multiple phases of a protocol with possible branches, it describes these phases as processes using the applied π-calculus.
Note that when ProVerif is run, these statements in the π calculus are transformed into Horn clauses.

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

W3C Working Draft

TABLE OF CONTENTS

§ 4.1. *PublicKeyCredential* Interface

The PublicKeyCredential interface inherits from Credential [CREDENTIAL-MANAGEMENT-1], and contains the attributes that are returned to the caller when a new credential is created, or a new assertion is requested.

```
[SecureContext]
interface PublicKeyCredential : Credential {
    [SameObject] readonly attribute ArrayBuffer              rawId;
    [SameObject] readonly attribute AuthenticatorResponse    response;
    [SameObject] readonly attribute AuthenticationExtensions clientExtensionResults;
};
```

id

This attribute is inherited from Credential, though PublicKeyCredential overrides Credential's getter, instead returning the base64url encoding of the data contained in the object's [[identifier]] internal slot.

rawId

This attribute returns the ArrayBuffer contained in the [[identifier]] internal slot.

***response*, of type AuthenticatorResponse, readonly**

This attribute contains the authenticator's response to the client's request to either create a public key credential, or generate an authentication assertion. If the PublicKeyCredential is created in response to create(), this attribute's value will be an AuthenticatorAttestationResponse, otherwise, the PublicKeyCredential was created in response to get(), and this attribute's value will be an AuthenticatorAssertionResponse.

***clientExtensionResults*, of type AuthenticationExtensions, readonly**

This attribute contains a map containing extension identifier → client extension output entries produced by the extension's client extension processing.

```
(*Registration*)
new Ns1:nonce;
new Ns2: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, Nt:nonce) = getmessAtt(m) in
let ver = checksignAtt(m, attpkU1) in

(*Second Registration *)
out(c, senc((Ns2,RP_id2),k));
in(c, s3:bitstring);
let m= sdec(s3,k) in
let (pkY2: pkey, attpkU2:AttestationPublicKey, credUser: bitstring, Nt:nonce) = getmessAtt(m) in
if (attpkU2<>attpkU1) then event reachSameKey(attpkU2,attpkU1);

new Ns11:nonce;
let bla= nonce_to_bitstring(Ns11) in
out(c,senc(bla,k))
.
process
  new k:key;
  new attskA:AttestationPrivatekey;
  let attpkA = spkAtt(attskA) in
  (
    !processUser( k, credentialID, attskA,attpkA) | !processServer(k, RP_id)
  )
```

Formal Verification

Security Properties

Threat Model : Network attacker

Security Properties :

- Integrity
- Authentication

ProVerif Description:

- *query attacker(attskA), query attacker(attpkA), query attacker(k),* and *query attacker(credentialID).*
- *sentChallengeResponse* and *validChallengeResponse.*

Privacy Properties

Threat Model : Web attacker

Security Properties : Unlinkability

ProVerif Description:

- Reachability event
- In ProVerif, we model privacy as the *event reachSameKey*, where we store key materials such as attestation keys for a given user in a table.

Results and next steps

A user (assuming TLS usage) can authenticate securely using cryptographic key material as their key material can not be violated by a network-level attacker.

The privacy property of unlinkability between origins can be violated

1. In self-attestation, the authenticator creates the attestation signature with his credential private key.
2. With elliptic curve direct anonymous attestation (ECDAA),an anonymous authentication credential is used to blindly sign the attestation credentials.
3. Authenticators of the same model could use the same attestation keys.

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.

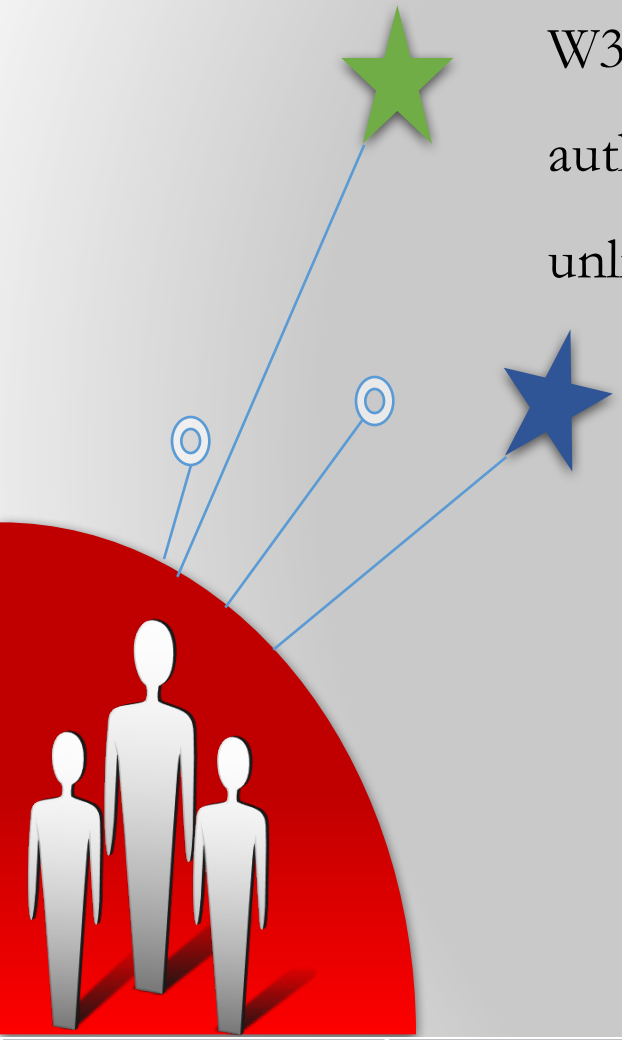
The server must only learn that their authenticator is trusted and not which particular authenticator is being used.

The authenticator contacts the issuer of attestation credentials (such as their device manufacturer) for their device and requests membership to a group.

The issuer grants the authenticator an attestation credential.

The host is now able to anonymously authenticate itself as a group member to a server.

Conclusion



W3C Web Authentication protocol is secure and so ensures strong user authentication, However it does allow violations of the privacy of users in terms of unlinkability of users between origins.

The solution: generates dynamically an attestation public key or use cryptographic techniques such as ECDAA or zero-knowledge proofs.

Thank you for your attention

"Cryptography is the ultimate form of non-violent direct action." **Julian Assange**

"Properly implemented strong crypto systems are one of the few things that you can rely on." **Edward Snowden**