Network Working Group
Internet-Draft
Intended status: Experimental
Expires: April 21, 2014

S. Farrell
Trinity College Dublin
P. Hoffman
VPN Consortium
M. Thomas
Phresheez

October 18, 2013

HTTP Origin-Bound Authentication (HOBA)
draft-ietf-httpauth-hoba-02

Abstract

HTTP Origin-Bound Authentication (HOBA) is a design for an HTTP authentication method with credentials that are not vulnerable to phishing attacks, and that does not require any server-side password database. The design can also be used in Javascript-based authentication embedded in HTML. HOBA is an alternative to HTTP authentication schemes that require passwords.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 21, 2014.

Copyright Notice

Table of Contents

1. Introduction

[[Commentary is in double-square brackets, like this. Fewer things this time.]]

HTTP Origin-Bound Authentication (HOBA) is an authentication design that can be used as an HTTP [RFC2616] authentication scheme and for Javascript-based authentication embedded in HTML. The main goal of HOBA is to offer an easy-to-implement authentication scheme that is not based on passwords, but that can easily replace HTTP or HTML forms-based password authentication. Deployment of HOBA can reduce or eliminate password entries in databases, with potentially significant security benefits.

HOBA is an HTTP authentication mechanism that complies with the framework for such schemes; see [RFC2617] for the original specification, and [I-D.ietf-httpbis-p7-auth] for clarifications and updates to the HTTP authentication framework. As a JavaScript design, HOBA demonstrates a way for clients and servers to interact using the same credentials that are used by the HTTP authentication scheme.

Current HTTP authentication methods (Basic and Digest), as well as username/password authentication using web forms, have been in use for many years, but being based on passwords, are susceptible to theft of server-side databases. Instead of passwords, HOBA uses digital signatures as an authentication mechanism. HOBA also adds useful features such as credential management and session logout. In HOBA, the client creates a new public-private key pair for each host ("web-origin" [RFC6454]) to which it authenticates. These keys are used in HOBA for HTTP clients to authenticate themselves to servers in the HTTP protocol or in a Javascript authentication program.

Session management with HOBA is identical to username/password session management. Typically, the session management tool (such as PHP, Python CGI, and so on) inserts a session cookie into the output to the browser. HOBA does nothing to help or hurt session cookie hijacking; TLS is still required for that.

HOBA keys are "bare keys", so there is no need for the semantic overhead of PKIX certificates, particularly with respect to naming and trust anchors. The client public key ("CPK") structures in HOBA do not have any publicly-visible identifier for the user who possesses the corresponding private key, nor the web-origin with which the client is using the CPK.

HOBA also defines some services that are required for modern HTTP authentication:

- Servers can bind a CPK with an identifier, such as an account name. Servers using HOBA define their own policies for binding CPKs with accounts during account registration.

- Users are likely to use more than one device or user agent (UA) for the same HTTP based service, so HOBA gives a way to associate more than one CPK to the same account, but without having to register for each separately.

- Users are also likely to lose a private key, or the client's memory of which key pair is associated with which origin, such as when a user loses the computer or mobile device in which state is stored. HOBA allows for clients to tell servers to delete the association between an existing CPK and an account.

- Logout features can be useful for UAs, so HOBA defines a way to close a current HTTP "session", and also a way to close all current sessions, even if more than one session is currently active from different UAs for the same account.

- Since there are always devices and applications in which state of the art digital signature mechanism runtimes are significant, and since HTTP authentication in theory requires that every HTTP request to a given realm have a signature in an "Authorization" header field, and since HOBA is a challenge response scheme, we also define a way in which HTTP servers can indicate the duration for which they will consider a given challenge value to be valid. As a consequence we also define a way for UAs to fetch a fresh challenge.

1.1. Interfacing to Applications (Cookies)

HOBA can be used as a drop-in replacement for password-based user authentication schemes used in common web applications. The simplest way in which this can be done is to (re-)direct the UA to a HOBA "Login" URL and for the response to a successful HTTP request containing a HOBA signature to set a session cookie [RFC6265]. Further interactions with the web application will then be secured via the session cookie, as is commonly done today.

While cookies are bearer tokens, and thus weaker than HOBA signatures, they are currently ubiquitously used. If non-bearer token session continuation schemes are developed in future in the IETF or elsewhere, then those can interface to HOBA as easily as with any password based authentication scheme.

1.2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

A client public key ("CPK") is the public key and associated cryptographic parameters needed for a server to validate a signature.

The term "account" is (loosely) used to refer to whatever data structure(s) the server maintains that are associated with an identity. That will contain of at least one CPK and a web-origin; it will also optionally include an HTTP "realm" as defined in the HTTP authentication specification. It might also involve many other non-standard pieces of data that the server accumulates as part of account creation processes. An account may have many CPKs that are considered equivalent in terms of being usable for authentication, but the meaning of "equivalent" is really up to the server and is not defined here.

When describing something that is specific to HOBA as an HTTP authentication mechanism or HOBA as a JavaScript implementation, this document uses the terms "HOBA-http" and "HOBA-js", respectively.

Web client: the content and javascript code that run within the context of a single UA instance (such as a tab in a web browser).

User agent (UA): typically, but not always, a web browser.

User: a person who is running a UA. In this document, "user" does not mean "user name" or "account name".

This specification uses the Augmented Backus-Naur Form (ABNF) notation of [RFC5234]

2. The HOBA Authentication Scheme

A UA that implements HOBA maintains a list of web-origins and realms. The UA also maintains one or more client credentials for each web-origin/realm combination for which it has created a CPK.

On receipt of a challenge (and optional realm) from a server, the client marshals an HOBA to-be-signed (TBS) blob that includes a client generated nonce, the web-origin, the realm, an identifier for the CPK and the challenge string; and signs that hashed blob with the private key corresponding to the CPK for that web-origin. The formatting chosen for this TBS blob is chosen so as to make server-side signature verification as simple as possible for a wide range of current server tooling.

Figure 1 specifies the ABNF for the signature input.

```
HOBA-TBS = nonce alg origin [ realm ] kid challenge
```

```
nonce = unreserved
alg = 1*2DIGIT
origin = scheme "://" authority ":" port
realm = unreserved
kid = unreserved
challenge = unreserved
```

Figure 1: To-be-signed data for HOBA

The fields above contain the following:

- nonce: is a random value chosen by the UA and MUST be base64url encoded before being included in the HOBA-TBS value. UAs MUST be able to use at least 32 bits of randomness in generating a nonce. UAs SHOULD be able to use 64 or more bits of randomness for nonces.

- alg: specifies the signature algorithm being used encoded as an ASCII character as defined in Section 9.3. RSA-SHA256 MUST be supported, RSA-SHA1 MAY be supported. The IANA registered algorithm values are encoded as ASCII numbers; for example, the encoding of RSA-SHA256 is 0x30.

- origin: is the web origin expressed as the concatenation of the scheme, authority and port from [RFC3986]. These are not base64 encoded as they will be most readily available to the server in plain text. For example, if accessing the URL "https://www.example.com:8080/foo" then the bytes input to the signature process will be "https://www.example.com:8080" There is no default for the port number, which MUST be present.

- realm: is similarly just a string with the syntactic restrictions defined in [I-D.ietf-httpbis-p7-auth]. If no realm is specified for this authentication then this is absent. (A missing field here is no problem since both sides know when it needs to be there.)

- kid: is a key identifier - this MUST be a base64url encoded value that is presented to the server in the HOBA client result (see below).

- challenge: MUST be a base64url encoded challenge value that the server chose to send to the client

The HOBA-TBS string is the input to the client's signing process, but is not itself sent over the network since some fields are already inherent in the HTTP exchange. The challenge however is sent over

the network so as to make it simpler for a server to be stateless. (One form of stateless challenge might be a ciphertext that the server decrypts and checks, but that is an implementation detail.) The value that is sent over the network is the HOBA "client result" which we now define.

The HOBA "client result" is a dot-separated string that includes the signature and is sent in the HTTP Authorized header field value using the value syntax defined in Figure 2. The "sig" value is the base64url encoded version of the binary output of the signing process. The kid, challenge and nonce are as defined above and are also base64url encoded.

```
HOBA-RES = kid "." challenge "." nonce "." sig
sig = unreserved
```

Figure 2: HOBA Client Result value

The HOBA scheme is far from new, for example, the basic idea is pretty much identical to the first two messages from "Mechanism R" on page 6 of [MI93] which predates HOBA by 20 years.

3. Introduction to the HOBA-http Mechanism

An HTTP server that supports HOBA authentication includes the "HOBA" auth-scheme value in a WWW-Authenticate header field when it wants the client to authenticate with HOBA. Note that the HOBA auth-scheme might not be the only one that the server includes in a WWW-Authenticate.

- If the "HOBA" scheme is listed, it MUST be followed by two or more auth-param values. The auth-param attributes defined by this specification are below. Other auth-param attributes MAY be used as well. Unknown auth-param attributes MUST be ignored by clients, if present.

- The "challenge" attribute MUST be included. The challenge is the string made up of the base64url encoded octets that the server wants the client to sign in its response. The challenge SHOULD be unique for every HTTP 401 response in order to prevent replay attacks from passive observers.

- An "expires" attribute MUST be included that specifies the number of seconds from the time the HTTP response is emitted for which responses to this challenge can be accepted.

- A "realm" attribute MAY be included to indicate the scope of protection in the manner described in HTTP/1.1, Part 7 [I-D.ietf-httpbis-p7-auth]. The "realm" attribute MUST NOT appear more than once.

When the "client response" is created, the UA encodes the HOBA client-result (a string matching the HOBA-RES production in Figure 2 as an auth-param with the name "result" and returns that in the Authorization header.

The server MUST check the cryptographic correctness of the signature based on a public key it knows for the kid in the signatures, and if the server cannot do that, or if the signature fails cryptographic checks, then validation has failed. The server can use any additional mechanisms to validate the signature. If the validation fails, or if the server chooses reject the signature for any reason whatsoever, the server aborts the transaction via a 403 Forbidden HTTP response.

Note that a HOBA signature is good for however long the expires attribute allows. This means that replay is possible within the time window specified by the "expires" value chosen by the server. Servers can attempt to detect any such replay and MAY react to such replays by responding with a second (or subsequent) 401-status HTTP response containing a new challenge.

UAs MAY optimise their use of challenges by pre-fetching a challenge value, for example after expires/2 seconds have elapsed, using the ".well-known/hoba/getChal" scheme described later in this document. This also allows for pre-calculation of HOBA signatures, if that is required in order to produce a responsive user interface.

4. Introduction to the HOBA-js Mechanism

Web sites using JavaScript can also perform origin-bound authentication without needing to involve the HTTP layer, and by inference not needing HOBA-http support in browsers. HOBA-js is not an on-the-wire protocol like HOBA-http is: instead, it is a design pattern that can be realized completely in JavaScript served in normal HTML pages.

One element is required for HOBA-js: localStorage (see http://www.w3.org/TR/webstorage/) from HTML5. is used for persistent key storage. For example, an implementation would store a dictionary account identifier, public key, private key tuples in the origin's localStorage for subsequent authentication requests. How this information is actually stored in localStorage is an implementation detail. This type of key storage relies on the security properties

of the same-origin policy that localStorage enforces. See the security considerations for discussion about attacks on localStorage.

Because of JavaScript's same-origin policy, scripts from subdomains do not have access to the same localStorage that scripts in their parent domains do. For larger or more complex sites, this could be an issue that requires enrollment into subdomains, which could be a hassle for users. One way to get around this is to use session cookies because they can be used across subdomains. That is, with HOBA-js, the user might log in using a single well-known domain, and then the server uses session cookies to navigate around a site.

Another element will be highly desirable for HOBA-js when it becomes available: WebCrypto (see http://www.w3.org/TR/WebCryptoAPI). In lieu of WebCrypto, JavaScript crypto libraries can be employed with the known deficiencies of PRNG, and the general immaturity of those libraries.

5. HOBA's Authentication Process

This section describes how clients and servers use HOBA for authentication. The interaction between an HTTP client and HTTP server using HOBA happens in three phases: the CPK preparation phase, the signing phase, and the authentication phase. This section also covers the actions that give HOBA similar user features as today's passwords have.

5.1. CPK Preparation Phase

In the CPK preparation phase, the client determines if it already has a CPK for the web-origin it is going to. If the client has a CPK, the client will use it; if the client does not have a CPK, it generates one in anticipation of the server asking for one.

5.2. Signing Phase

In the signing phase, the client connects to the server, the server asks for HOBA-based authentication, and the client authenticates by signing a blob of information as described in the previous sections.

5.3. Authentication Phase

The authentication phase is completely dependent on the policies and practices of the server. That is, this phase involves no standardized protocol in HOBA-http; in HOBA-js, there is no suggested interaction template.

In the authentication phase, the server extracts the CPK from the signing phase and decides if it recognizes the CPK. If the server recognizes the CPK, the server may finish the client authentication process.

If this stage of the process involves additional information for authentication, such as asking the user which account she wants to use (in the case where a UA is used for multiple accounts on a site), the server can prompt the user for account identifying information or the user could choose based on HTML offered by the server before the 401 is triggered. None of this is standardized: it all follows the server's security policy and session flow. At the end of this, the server probably assigns or updates a session cookie for the client.

During the authentication phase, if the server does not recognize the CPK, it could use HTML and JavaScript to ask the user if they are really a new user or want to associate this new CPK with an already-joined CPK. The server can then use some out-of-band method (such as a confirmation email round trip, SMS, or an UA that is already enrolled) to verify that the "new" user is the same as the already-enrolled one. Thus, logging in on a new user agent is identical to logging in with an existing account.

If the server does not recognize the CPK the server might send the client through a either a join or login-new-UA (see below) process. This process is completely up to the server, and probably entails using HTML and JavaScript to ask the user some questions in order to assess whether or not the server wants to give the client an account. Completion of the joining process might require confirmation by email, SMS, Captcha, and so on.

Note that there is no necessity for the server to initiate a joining or login process upon completion of the signing phase. Indeed, the server may desire to challenge the UA even for unprotected resources and set a session cookie for later use in a join or login process as it becomes necessary. For example, a server might only want to offer an account to someone who had been to a few pages on the web site; in such a case, the server could use the CPK from an associated session cookie as a way of building reputation for the user until the server wants the user to join.

6. Other Parts of the HOBA Process

The authentication process is more than just the act of authentication. In password-based authentication and HOBA, there are other processes that are needed both before and after an authentication step. This section covers those processes. Where possible, it combines practices of HOBA-http and HOBA-js; where that is not possible, the differences are called out.

All additional services MUST be performed in TLS-protected sessions ([RFC5246]). If the current HTTP traffic is not running under TLS, a new session is started before any of the actions described here are performed.

HOBA-http uses a well-known URL [RFC5785] "hoba" as a base URI for performing many tasks: "https://www.example.com/.well-known/hoba". These URLs are based on the name of the host that the HTTP client is accessing. There are many use cases for these URLs to redirect to other URLs: a site that does registration through a federated site, a site that only does registration under HTTPS, and so on. Like any HTTP client, HOBA-http clients MUST be able to handle redirection of these URLs. [[There are a bunch of security issues to consider related to cases where a re-direct brings you off-origin.]]

6.1. Registration

Normally, a registration (also called "joining") is expected to happen after a UA receives a WWW-Authenticate for a web-origin and realm (for HOBA-http) or on demand (for HOBA-js) for which it has no associated CPK. The process of registration for a HOBA account on a server is relatively light-weight. The UA generates a new key pair, and associates it with the web-origin/realm in question.

Note that if the UA has a CPK associated with the web-origin, but not for the realm concerned, then a new registration is REQUIRED. If the server did not wish for that outcome, then it ought to use the same or no realm.

The registration message for HOBA-http is sent as a POST message to the URL ".well-known/hoba/register" with an HTML form (x-www-form-encoded) described below; The registration message for HOBA-js can be in any format specified by the server, but it could be the same as the one described here for HOBA-http. It is up to the server to decide what kind of user interaction is required before the account is finally set up.

The registration message sent to server has one mandatory field (pub) and some optional fields that allow the UA to specify the type and value of key and device identifiers that the UA wishes to use.

- pub: is a mandatory field containing the PEM formatted public key of the client. See Appendix C of [RFC6376] for an example of how to generate this key format.

- kidtype: contains the type of key identifier, this is a numeric value intended to contain one of the values from Section 9.4. If this is not present then the mandatory to implement "DANE-hash" option MUST be used.

- kid: contains the key identifier as a base64url encoded string that is of the type indicated in the kidtype. If the kid is a hash of a public key then the correct (base64url encoded) hash value MUST be provided and the server SHOULD check that and refuse the registration if an incorrect value was supplied.

- didtype: specifies a kind of device identifier intended to contain one of the values from Section 9.5, if absent then the "string" form of device identifier MUST be used.

- did: a UTF8 string that specifies the device identifier. This can be used to help a user be confident that authentication has worked, e.g., following authentication some web content might say "You last logged in from device 'did' at time T."

- curtime: the number of milliseconds since the Unix epoch, formatted as [[not yet specified]]. This is based on the expectation that the synchronization between the browser's and server's clocks is sufficiently reliable. This is used to guard against the replay of a legitimate registration message. The server needs to have a replay cache covering a signature timeout window. This might be done using a database table that is keyed (in the database sense of the term) using the signature bits. If the signature is in the replay table, the server will reject it. If the timestamp in the signature is outside the current replay cache window then it also gets rejected.

[[An addition of the ability for the server to reject a client with potential time skew and give it a nonce (as with HOBA-http) would allow the size of the replay cache to be set to just a few minutes rather than a much longer period. Or the HOBA server could always use a nonce method. This is worthy of more discussion.]].

[[A comment raised on the list was: how does the UA know that the registration process has completed successfully or badly? That's not yet handled.]]

6.2. Associating Additional Keys to an Exiting Account

6.2.1. Moving private keys

It is common for a user to have multiple UAs, and to want all those UAs to be able to authenticate to a single account. One method to allow a user who has an existing account to be able to authenticate on a second device is to securely transport the private and public keys and the origin information from the first device to the second. If this approach is taken, then there is no impact on the HOBA-http or HOBA-js so this is a pure UA implementation issue and not discussed further.

6.2.2. One time password

To enroll a new UA using an existing UA, the user requests a one-time password that will be entered in the new UA. On HOBA-http, this is done with the URL ".well-known/hoba/associate-start" using a POST message [[more detail is clearly needed here]]. The server displays a one-time password that expires in a short amount of time (such as 30 minutes) on the currently enrolled UA. The new UA generates a new CPK and sends it to the server. In HOBA-http, this is done with a POST to ".well-known/hoba/associate-finish" [[more detail needed here]].

The one-time password should be easy for a user to type into the new UA, but at the same time needs to have enough randomness to prevent an attacker from succeeding if they try to use the password during its short lifetime. After the server gets the password in the new UA, it verifies the signature and verifies that the password supplied is in a list of unexpired one-time passwords. If so, the server knows that the authenticated user is associated with the second CPK. The server can choose to associate the two CPKs with one account. Whether to do so is entirely at the server's discretion however, but the server needs make the outcome clear to the user.

Alternatively, if an already-enrolled UA is not available to the user when they want to associate a new UA with an existing account, and the site has an out-of-band communication mechanism such as email or SMS associated with that account, the user can request that a one-time password be sent to the user out of band. The user receives the one-time password and, using the new UA, tells it to the server. On HOBA-http, this is done with the URL ".well-known/hoba/associate-from-oob" using a POST message [[more detail is clearly needed here]].

[[Note: the hoba.ie implementation doesn't do this - I omitted a human memorable OTP for now. The reason is I think the probability that spammers would try those is too high. So maybe we should delete this entirely?]]

6.2.3. Out of band

There are various other ways in which a server can provide ways to allow a new UA to be registered to the same account. For example, simply providing a web page where, using the first UA, the user can generate a URL (containing some cryptographic value) that the user then later de-references on the second UA would suffice. The user could have e-mail that URL to herself for example, of the web server accessed at the first UA could automatically do that.

[[Need to think about how much detail to put in here. Its all fairly obvious, but we probably should talk about the security considerations. The hoba.ie site btw has a few of these methods implemented.]]

6.3. Logging Out

The user can tell the server it wishes to log out. With HOBA-http, this is done by going to the URL ".well-known/hoba/logout". The UA SHOULD also delete or modify session cookies associated with the session so that the user's state is no longer "logged in."

The server MUST NOT allow TLS session resumption for any logged out session. [[Need to determine if this is needed or even a good idea.]]

The server SHOULD also revoke or delete any cookies associated with the session.

6.4. Getting a Fresh Challenge

The UA can get a "fresh" challenge from the server. In HOBA-http, it sends a POST message to ".well-known/hoba/getchal". If successful, the response response MUST include a fresh (base64url encoded) HOBA challenge for this origin in the body of the response.

7. Mandatory-to-Implement Algorithms

RSA-SHA256 MUST be supported. RSA-SHA1 MAY be used. RSA modulus lengths of at least 2048 bits SHOULD be used.

8. Security Considerations

If key binding was server-selected then a bad actor could bind different accounts belonging to the user from the network with possible bad consequences, especially if one of the private keys was compromised somehow.

Binding my CPK with someone else's account would be fun and profitable so SHOULD be appropriately hard. In particular the string generated by the server MUST be hard to guess, for whatever level of difficulty is chosen by the server. The server SHOULD NOT allow a random guess to reveal whether or not an account exists.

[[The potential impact on privacy of HOBA needs to be addressed. If a site can use a 401 and a CPK to track users without permission that would be not-so-nice so some guidance on how a UA could indicate to a user that HOBA stuff is going on might be needed.]]

[[Lots more TBD, be nice to your private keys etc. etc.]]

8.1. localStorage Security for Javascript

Our use of localStorage will undoubtedly be a cause for concern. localStorage uses the same-origin model which says that the scheme, domain and port define a localStorage instance. Beyond that, any code executing will have access to private keying material. Of particular concern are XSS attacks which could conceivably take the keying material and use it to create UAs under the control of an attacker. But XSS attacks are in reality across the board devastating since they can and do steal credit card information, passwords, perform illicit acts, etc, etc. It's not clear that we introduce unique threats from which clear text passwords don't already suffer.

Another source of concern is local access to the keys. That is, if an attacker has access to the UA itself, they could snoop on the key through a javascript console, or find the file(s) that implement localStorage on the host computer. Again it's not clear that we are worse in this regard because the same attacker could get at browser password files, etc too. One possible mitigation is to encrypt the keystore with a password/pin the user supplies. This may sound counter intuitive, but the object here is to keep passwords off of servers to mitigate the multiplier effect of a large scale compromise ala LinkedIn because of shared passwords across sites.

It's worth noting that HOBA uses asymmetric keys and not passwords when evaluating threats. As various password database leaks have shown, the real threat of a password breach is not just to the site that was breached, it's all of the sites a user used the same password on too. That is, the collateral damage is severe because password reuse is common. Storing a password in localStorage would also have a similar multiplier effect for an attacker, though perhaps on a smaller scale than a server-side compromise: one successful crack gains the attacker potential access to hundreds if not thousands of sites the user visits. HOBA does not suffer from that

attack multiplier since each asymmetric key pair is unique per site/ UA/user.

8.2. Multiple Accounts on One User Agent

A shared UA with multiple accounts is possible if the account identifier is stored along with the asymmetric key pair binding them to one another. Multiple entries can be kept, one for each account, and selected by the current user. This, of course, is fraught with the possibility for abuse, since a server is potentially enrolling the device for a long period and the user may not want to have to be responsible for the credential for that long. To alleviate this problem, the user can request that the credential be erased from the browser. Similarly, during the enrollment phase, a user could request that the key pair only be kept for a certain amount of time, or that it not be stored beyond the current browser session.

9. IANA Considerations

9.1. HOBA Authentication Scheme

Authentication Scheme Name: HOBA

Pointer to specification text: [[this document]]

Notes (optional): The HOBA scheme can be used with either HTTP servers or proxies. [[Still need to figure out what it means to do this through proxies.]]

9.2. .well-known URLs

We probably want a new registry for the labels beneath .well-known/ hoba so that other folks can add additional features in a controlled way, e.g. for CPK/account revocation or whatever.

9.3. Algorithm Names

TBD, hopefully re-use and existing registry

"0" means RSA-SHA256

"1" means RSA-SHA1

9.4. Key Identifier Types

"0" means a hashed public key, as done in DANE. [RFC6698]

"1" means a URI, such as a mailto: or acct: URI, but anything conforming to [RFC3986] is ok.i

"2" means an unformatted string, at the user's/UA's whim

9.5. Device Identifier Types

"0" means an unformatted nickname, at the user's/UA's whim

10. Implementation Status

[[Note to RFC editor - please delete this section before publication.]]

This section records the status of known implementations of the protocol defined by this specification at the time of posting of this Internet-Draft, and is based on a proposal described in [RFC6982]. The description of implementations in this section is intended to assist the IETF in its decision processes in progressing drafts to RFCs. Please note that the listing of any individual implementation here does not imply endorsement by the IETF. Furthermore, no effort has been spent to verify the information presented here that was supplied by IETF contributors. This is not intended as, and must not be construed to be, a catalog of available implementations or their features. Readers are advised to note that other implementations may exist.

According to [RFC6982] "this will allow reviewers and working groups to assign due consideration to documents that have the benefit of running code, by considering the running code as evidence of valuable experimentation and feedback that has made the implemented protocols more mature. It is up to the individual working groups to use this information as they see fit".

At the time of writing there are two known implementations. One done by Stephen Farrell of HOBA-HTTP and a HOBA-JS variant implements the -01 version of HOBA and is available from https://hoba.ie/ which site also includes a demonstration of HOBA.

There is another implementation by Michael Thomas of an HOBA-JS variant.

11. Acknowledgements

Thanks to the following for good comments received during the preparation of this specification: Julian Reschke, James Manger, Michael Sweet [[and many more to be added]]. All errors and stupidities are of course the editors' fault.

12. References

12.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[RFC2617] Franks, J., Hallam-Baker, P.M., Hostetler, J.L., Lawrence, S.D., Leach, P.J., Luotonen, A., and L. Stewart, "HTTP Authentication: Basic and Digest Access Authentication", RFC 2617, June 1999.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

[RFC5234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

[RFC5785] Nottingham, M. and E. Hammer-Lahav, "Defining Well-Known Uniform Resource Identifiers (URIs)", RFC 5785, April 2010.

[RFC6454] Barth, A., "The Web Origin Concept", RFC 6454, December 2011.

[RFC6698] Hoffman, P. and J. Schlyter, "The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA", RFC 6698, August 2012.

12.2. Informative References

[I-D.ietf-httpbis-p7-auth] Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Authentication", draft-ietf-httpbis-p7-auth-22 (work in progress), February 2013.

[MI93] Mitchell, and Thomas, "Standardising Authentication Protocols Based on Public-Key Techniques.", Journal of Computer Security 2 (1993): 23-36. , 1993.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, October 2006.

[RFC6265] Barth, A., "HTTP State Management Mechanism", RFC 6265, April 2011.

[RFC6376] Crocker, D., Hansen, T., and M. Kucherawy, "DomainKeys Identified Mail (DKIM) Signatures", STD 76, RFC 6376, July 2013.

[RFC6982] Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code: The Implementation Status Section", RFC 6982, July 2013.

Appendix A. Problems with Passwords

By far the most common mechanism for web authentication is passwords that can be remembered by the user, called "memorizable passwords". There is plenty of good research on how users typically use memorizable passwords ([[handful of citations goes here]]), but some of the highlights are that users typically try hard to reuse passwords on as many web sites as possible, and that web sites often use either email addresses or users' names as the identifier that goes with these passwords.

If an attacker gets access to the database of memorizable passwords, that attacker can impersonate any of the users. Even if the breach is discovered, the attacker can still impersonate users until every password is changed. Even if all the passwords are changed or at least made unusable, the attacker now possesses a list of likely username/password pairs that might exist on other sites.

Using memorizable passwords on unencrypted channels also poses risks to the users. If a web site uses either the HTTP Plain authentication method, or an HTML form that does no cryptographic protection of the password in transit, a passive attacker can see the password and immediately impersonate the user. If a hash-based authentication scheme such as HTTP Digest authentication is used, a passive attacker still has a high chance of being able to determine the password using a dictionary of known passwords.

[[Say a bit about non-memorizable passwords. Still subject to database attack, although that doesn't give the attacker knowledge for other systems. Safe if digest authentication is used, but that's rare.]]

Appendix B. Examples

[[Will add more later and probably use example.org. Note these still use the -01 version of the "origin" abnf production.]]

The following values show an example of HOBA-HTTP authentication to the origin https://hoba-local.ie. Carriage-returns have been added and need to be removed to validate the example.

```
-----BEGIN PRIVATE KEY-----
MIIEvwIBADANBgkqhkiG9w0BAQEFAASCBKkwggSlAgEAAoIBAQDNF6tZUbsM7ZrO
5Lyzvn15lJAfOz7j7xdc3hmeSOfh/DCiJWwE5qqffrOvOvXYN+qUTlsXPeBYdrz/
my0YYC02u2QFhDbFRvpM/EuMzZUWTQzkKyU7nSjtPqlLZJJ/Rh6PnjTgImoIMn92
JZgJZgl/vzd29K5Z94JzdJ4Z5bNmQ/gCpjlv0Wvi+GpJ3lDo1csDEyxATxyTUx1J
K73+/RPoNgUF9nrXN6kqeH7RkERRz+PFhfGM6r3tU5povJxOP0bzVl6R4kptWLn2
GqDb5LS9iwzsk6YHWQcLOIAMVZ9ITPN2PSuQX0ym81B9qB1V6fTfo2r0Yo1Djn6C
YRX62p4dAgMBAAECggEBAJiyWLcFrPhxJ2OG1iAVYaJVxAAcwjQ+XOydyAEbUtnk
Q+lVZ1k2zC43zVxXz5aN+y80L4ncXd4/eXPteuO9J6yqVEvvJkA3GkCbTzykC64w
67otjWkXF9ObZbxmQtRTxokzRzbhKIS15ER4tPu6ZrQgEBGXFwCQ0SVY3CV36dvm
xU2Y90wf98BGQF5VO47S9h8N/VtBn1ttI/6BhQHGsM+TbRR0b6oT9Mp/kqh5jZ2P
dH/l7Q6aj6zeFYljvI6HUpjc0mVOSKPAO0qYLZNYC0sqXxW9vCtEODziFZwrO2tL
jm14/H8+AZvtR6ZvbzP8h3G95PNdG4Z1eGZv5pOoJoECgYEA8omsyssjVgLIon60
Q6h3EjOi8I+mfvXYXcAZhv/1MnojFZgMLmuZ13Lgi3dIHYFNYGFRi8g9iYU/Ljtf
G3B9o6PQsvL0yym85TZzF/yn+kbinHR2yQUiA1Ck1rnrFx8dla/BjAplPxumyh/z
HyNFplVbOeRQS4K5F72wScuB9DECgYEA2Hnq1OSLetl4lrGAEHDoiIKw++ACQ/Tn
w5O+xAxOXr9iOmczwpHls+zFSQHty4za8C/tBLxg6HCHKZSO4lh+p5SR21UNgwfr
X6xBZ4xoH9zHTFdD2yrHdxAoItdT5oMTAUNHxDS8kcSY1+YcjRs2T4kcONIxeBlp
O++dxK9I6a0CgYEAp52WGSCCbzLFTeea1RdcEuw0s2PTgPKOcVwNSEskPZpDHO1T
ndEnJMpzfG8XG6z8uJsJLD1aqeu4Wk8Vz3TSn4Da/pEBtFZIAXC74dvuivzqJ44l
eY9ejkPxZ6RdYEFUxNoOPKYCiralchLahq5tuCJNRZkQFN9m441og9dtHEECgYEA
qnvRqlpXUqfEZYFi5w/UwfWTJrozbouInylTOpiqe8njtTUjuV8ndPzKHoYrXXwP
zMshsfIdq9E7UU7S/IVPMfE6sW6ZVpE9GDrTw5X7RuSb/I5ZPVjCgA00XsQQKmEd
7YesFGSoAXDAIn/yClrc+eR0WneHSBtTGkXKjWSyWn0CgYAlHdG3wj/sOL+JI7p8
fgSoRpkrbjfrczqSCDVFbT84sUz3dXObZecrI3v0XUZySakIDha5ZbxXNST/bfea
kReKsoeFY9J7os0hbfGwxiCQwD3wN9tS2yJbJYl1mCFc45YfylkXBzB0S2V/i6rv
vCWW1nijRb+WhpUF8HyKW64bxA==
-----END PRIVATE KEY-----
```

Key Identifier: Zhh5vD5ovE0NqTDOufSUFRu5dzZMe-KOrMX2cN3vqWw

Challenge: zzrYL7BaOQtlzOsl4fMY+EYcG4eT2h+JXi+jEGzozQ0=

Nonce: xXSFdZ-7ahM

Tbsorigin: httpshoba-local.ie443

The resulting signature is:

r1ZXAWPXpzkd9iyI9TvwNYb0LT6Nth4WRYL4ciLZD6Wvvsni8AYLduUEPdo5ezfo
K__W_Hi4nyHmtRzPpAW9YSGhsyYOd7GSZH7Kd6ncCPVBQuHQdHI5n6OJslitD7hK
t4bCtP3zGxkg_W71KGU2RXcQDfcTNmFcs2ice8RrrvNh1lzRViHO-scV0VNBk19J
LTUyaisiNKq-sNK14_RIG6AeivAGxLlXnN_RzttNe5d0XrXJ1nRUSFmeN6ZfVHE7
qf6lORqaMeyqsDoJe1MIyISn6sCGzMZmplizNw_eg2QJIJX3Txat9mTfT5UZYyUq
8meaqRXMhoQWHGLweFTNYw

The final HTTP header field sent with a request is then:

Authorization: result="Zhh5vD5ovE0NqTDOufSUFRu5dzZMe-KOrMX2cN3vq
Ww.zzrYL7BaOQtlzOsl4fMY+EYcG4eT2h+JXi+jEGzozQ0=.xXSFdZ-7ahM.r1ZX
AWPXpzkd9iyI9TvwNYb0LT6Nth4WRYL4ciLZD6Wvvsni8AYLduUEPdo5ezfoK__W
_Hi4nyHmtRzPpAW9YSGhsyYOd7GSZH7Kd6ncCPVBQuHQdHI5n6OJslitD7hKt4bC
tP3zGxkg_W71KGU2RXcQDfcTNmFcs2ice8RrrvNh1lzRViHO-scV0VNBk19JLTUy
aisiNKq-sNK14_RIG6AeivAGxLlXnN_RzttNe5d0XrXJ1nRUSFmeN6ZfVHE7qf6l
ORqaMeyqsDoJe1MIyISn6sCGzMZmplizNw_eg2QJIJX3Txat9mTfT5UZYyUq8mea
qRXMhoQWHGLweFTNYw"

Authors' Addresses

Stephen Farrell
Trinity College Dublin
Dublin 2
Ireland

Phone: +353-1-896-2354
Email: stephen.farrell@cs.tcd.ie

Paul Hoffman
VPN Consortium

Email: paul.hoffman@vpnc.org

Michael Thomas
Phresheez

Email: mike@phresheez.com