Tech Summit 10-2010 Notes

last edited by Jesse Stay 2 days ago

OpenID Technology Summit

November 1, 2010 - Facebook Headquarters

Background

OpenID Technology Summits are an important way for the OpenID Foundation to educate, as well as collaborate between members in person regarding important specifications and protocol discussions of concern to Foundation members. Members meet several times a year in various settings throughout the world, and meetings are usually hosted by Corporate representatives willing to contribute a location for members to meet. (How do I host a Summit?)

November's OpenID Technology Summit was hosted at Facebook Headquarters, and brought together discussions around the OpenID Connect and Artifact Binding proposals. The OpenID Foundation would like to thank Facebook for their generous contribution of hosting facilities, as well as the delicious lunch provided by the Company.

How Do I Participate?

Don Thibeau, (Executive Director) of the OpenID Foundation, reminded attendees that there are no free lunches. Attendees are encouraged to join the Foundation with a simple donation. If you are interested in joining the Foundation and contributing to this cause, please visit the Foundation's website and decide how you would like to contribute. The Foundation welcomes and relies on both corporate and individual members and participants, membership allowing those to both vote for, as well as to participate in leadership in the organization. 2011 Board elections are coming up, so be sure to register, nominate yourself if you're interested, and vote!

Agenda and Terms

The OpenID Technology Summit was agreed to be a wide open discussion where participating parties could openly discuss open issues at hand with the next era of OpenID technology. Topics proposed to discuss were:

- Discovery
- Signatures and Encryption
- Three similar OpenID Connect Proposals by Facebook, Google, and John Bradley
- Backwards Compatibility Strategies and Transition plans for future specs
- v.Next
- User information and the extensibility of user information
- Marketing of OpenID specifications and protocols - who to get information into the wild through user adoption and commitment
- LOA, PAPE, PXI
- Mobile, Desktop, and Rich UI experiences with OpenID
- Best Practices to Bootstrap OAuth 2.0
- Lightning updates of what people are doing in this space (this was decided to do over lunch)

While not all would be covered, it was decided to break out into a couple sessions at a time, beginning with a discussion on the three OpenID Connect proposals that were brought to the Summit. The Summit broke out into the following sessions:

OpenID Connect Proposals

Summary

The purpose of OpenID Connect is to solve 2 basic problems:

1. To identify individuals (authentication) - this is solved today by OpenID 2.0
2. To allow developers to call information about individuals (authorization) - this is solved today by OAuth 2.0

The problem is that while OpenID and OAuth attempt to solve 2 distinct problems, most developers really have a need for both, and are often using OAuth and variations of the spec to attempt to resolve both problems. To solve this, it has been proposed that a variation of OAuth be created, which places OpenID on top of OAuth, allowing distributed identification of individuals, while still enabling authorization, allowing developers to retrieve access to information about those individuals.

The goal of the various OpenID Connect proposals is to bring OpenID and OAuth together in a secure and simple manner which doesn't involve too much fancy encryption on the side of the developer.

3 basic proposals were brought to the group. Of the 3 proposals, a common theme came across. The client would request an openid identifier to the OAuth Endpoint*. The OAuth Endpoint recognizes the openid identifier and returns back a JSON-structured token identifying the user and providing the proper access token to authorize data for that user.

The differences between the proposals centered around how the data in the JSON objects were structured, as well as how user ids returned in the object were identified. These are the main differences between the 3 specs:

How do I figure out to the OAuth Endpoint URL?

In normal OpenID a user enters a specific URL, such as http://oidf.org, and on oidf.orga little HTML tag points any server trying to get a user's ID to the URL of the server that can identify that individual. In OpenID Connect, this process is similar. The user would enter a URL, e-mail address, or maybe even click on a pre-defined button that provides similar information, and the developer's code will follow the URL chosen by the user to look for a defined file in JSON format that tells the developer's code where to redirect the user to authenticate. This URL is the OAuth Endpoint URL, and should return the data proposed from the OpenID Technology Summit.

Proposal #1- Breno de Medeiros, Google

Proposal 1's main difference is the desire to have the user identifier stored in one field in the returned JSON object, in the form of a globally unique identifier prefixed to a "@domainname" format. (So, something like 1234567@google.com, for instance)

Proposal #2- David Recordon, Facebook

For Proposal 2, the user identifier stored in the JSON response is split into 2 separate fields in the response. One of those parameters being the globally unique identifier and the other being the domain of the service returning the object.

Proposal #3- John Bradley

Proposal 3 was very similar in nature to 1 and 2. The main difference being that returned tokens are signed and encrypted JSON objects containing a minimum of the user_id and an optional display_name parameter. This means, assuming you just need basic user data, the server does not have to go back and make an additional call to the authorization end point (Google, Facebook) to retrieve that data.

Consensus

The final consensus, after a discussion on SalesForce.com's use of OpenID, was to use the proposed format of proposal 2, encouraging the separation of the globally unique identifier and domain name as 2 distinctly separate parameters. The format of {12345,"twitter.com"} was suggested as the format for those parameters. Also, the return of the encapsulated JSON object (signed and encrypted) with user data already in it as the actual token was agreed upon.

Signatures and Encryption Proposals

A large concern of the current OAuth 2.0 spec is that access tokens and ids of users are completely visible in the browser, and can easily be taken and abused. Part of the goal of OpenID Connect is to find a secure, simple method to make this man-in-the-middle retrieval of data much harder, meeting standard Levels of Assurance so even some of the more secure organizations can use it.

The entire time of the discussion surrounding the signatures was focused on how to organize the signature, how to encrypt the signature, and what information to include in the signature. The discussion was taken to IIW the following week, and the following specs where drafted:

- JSON Token Spec Results
- JSON Token Encryption Spec Results
- JSON Token Naming Spec Results
- JSON Public Key Spec Results

Discovery Discussion

An important piece to the OpenID Connect process is how the Client (say, someone like Facebook.com or Google.com or similar where you would login) knows where to send the user in order to authenticate them before getting the permission that client needs to get more information about that user. Let's say that person wants their profile, contact, and stream information stored on Google, for instance. Somehow that user has to identify that Google is the place they want you to retrieve their data, and you, the Client (again, the place where they're trying to login or authenticate), have to know to go there to get it for them.

Issues

- **The NASCAR Problem** - on race cars, brands are placed prominently throughout the car for marketing and branding purposes. As OpenID becomes more prominent, websites have the potential to become similar - button after button of branded logins for the user to choose from (and this is already the case on many websites). OpenID Connect ought to work to make this a simpler case by removing the need for so many buttons, allowing the user to choose how they

want to log in to your site. It was discussed that OpenID Connect should remain flexible, allowing for the user to enter a URL, like existing OpenID implementations, or an e-mail address using technologies like WebFinger to identify the user. At the same time, sites can still choose to add their own buttons for users to choose from of commonly identifiable brands the user might already be associated with (A Google button, a Facebook button, a Yahoo button).

- **Process of Discovery, Host Meta DNS Poisoning** - Current OpenID methods allow the user to specify on a website somewhere (or the website can do this for them) where they want their authentication to take place. Some how the place they are logging in has to understand where to go to authenticate that user and identify them as a real person. It was discussed how this process of Discovery would take place. It was also discussed how the data would be returned (JSON or XML or existing HTML). In addition, in the process of discovery, how do you prevent the user from tricking the client into going to the wrong place to get their discovery information (allowing the user to authenticate themselves and return fake information back to the Relying Party)?

Consensus

- **NASCAR Problem** - It was decided to keep OpenID Connect flexible. It can solve the NASCAR problem (to an extent), or sites can choose to add as many buttons as they like, using OpenID Connect as the underlying standard to authenticate that user.
- **Discovery** - It was suggested, and generally agreed upon, that the Client (where the user logs in) would look at a Host Meta file to find out where to go authenticate the user (See Phil Windley's explanation here for a great explanation of Host Meta). It was also suggested that Host Meta files would be provided in a new JSON format (the previous standard being in XML).
- **DNS Poisoning** - No real agreement came together on a good solution to prevent the DNS poisoning problem for Host-meta mentioned above. It was suggested an ID and Secret of some sort be passed to ensure authenticity of the client and the location of the Host Meta file. Discussions were to be continued at IIW.

Profiles and Attributes Discussion

As part of the Discovery session, it was discussed the format of profile data returned in the JSON object. Google, Microsoft, MySpace, and others have adopted the Portable Contacts standard to present this data, while Facebook, Twitter, and many others have adopted other ways to present profile data for the user. The goal of the discussion was to come to a common consensus on the standard for presenting profile data to the developer in an extensible way. Joseph Smarr of Google and Tantek Çelik of Mozilla Foundation guided the discussion.

Several profile standards were discussed:

- **vCard** - vCard 3 eventually forked to vCard 4. The vCard 4 standard group was currently meeting at the time, discussing how to develop the standard, which is based on XML mapping. Apple and some enterprise consultancy groups are supporting the vCard 4 standard (will be part of the next version of MacOS).
- **OpenSocial** - OpenSocial was proposed as a standard to help standardized contact data. Eventually OpenSocial evolved towards the Portable Contacts standard (or PoCo), which is becoming the de facto standard for many companies including Mozilla and those mentioned above.
- **OpenID sreg** - sreg is a simpler protocol. It only has 7 fields, and is a subset of Portable Contacts. The standard is currently parked.
- **OpenID AX+Schema 1.0** - this is a generic framework, with no endpoint to set the data. There is no one working to evolve this schema - the group agreed to call it "parked".

While no official solution was agreed upon, it was agreed that the Portable Contacts community would become the end point to discuss user-related issues surrounding profiles and attributes. The goal of this was to hopefully come to a consensus within the Portable Contacts community on how the standard could evolve to satisfy the concerns of all parties involved.

It was also agreed that Portable Contacts would most likely not solve all issues, such as the case where small amounts of data are returned. The suggestion was to encourage the default of Portable Contacts, and then adapt your own data and formats when Portable Contacts wouldn't work for your situation.

The following action items came out of the discussion:

- Those at the meeting were encouraged to sign up for the Portable Contacts mailing list, and to encourage others to do so. This would provide a common place to solve all the profile-related problems.
- It was suggested that the Portable Contacts group meet together and come up with something that everyone else can agree upon. The goal would then be to sell the rest of the community.
- Joseph Smarr said he would e-mail the Portable Contacts list, and e-mail the OpenID specs list asking them to join the Portable Contacts list.
- Places to converge surrounding Portable Contacts include:
    - PortableContacts.net
    - portablecontacts.googlegroups.com
    - FederatedSocialWeb.net

Levels of Assurance Discussion

Levels of Assurance (also known as LOA) are government-mandated standards defining how secure a protocol is. As part of this discussion, the hope was to decide how OpenID Connect could meet the various LOA requirements.

Consensus

It was agreed that OpenID Connect could meet LOA levels 2 and 3 in varying ways. The ways to achieve full LOA certification are:

- To meet LOA 2 status, OpenID Connect would need to be implemented in a web server flow, away from the client (the browser).
- To meet LOA 3 status, OpenID Connect would need to be implemented in a web server flow, but with a web signature.
- It was debated whether LOA 4 was even worth attempting.
- The current client credential methods, such as that which Facebook and others are using, would not pass LOA level 1, but could achieve LOA level 2 once they move to an encrypted token.

The group also discussed the benefit to having the user pre-identified through the browser (like Chrome), allowing the browser itself to provide the identity of the user.

November 3, 2010

JSON Token Spec Results at IIW on Tuesday

We held two back-to-back sessions at IIW on Tuesday intended to produce consensus positions on JSON Web Tokens [http://self-issued.info/?p=349] , including signing and encryption. Substantial consensus emerged, which is described in the notes below.

These consensus decisions were in place by the start of the session:

- There is an envelope (a.k.a. header) that completely describes the cryptographic algorithm(s) used
- There is a payload (a.k.a. body) that is distinct from the envelope
- There is a signature that is distinct from the envelope and payload
- Base64url encoding without padding is used to encode the parts above
- The compact token representation separates the three encoded parts above by periods
- No line breaks or other whitespace may be present in this representation
- Encryption must be supported as well as signatures
- The token representation should be compact
- In particular, this means that multiple base64url encodings of the same content should be avoided
- Any need for canonicalization should be avoided

Open issues identified at the start of the session were:

- Ordering of the fields
- Ordering of the signing and encryption operations
- What can be in an envelope (a small fixed set of things or is it extensible)?
- What to sign (envelope.payload or just payload)?
- What can be in the payload (only sets of JSON token claims or arbitrary base64url encoded byte streams)?
- Do we need to support multiple signatures?
- Should we specify a JSON serialization as well as a compact serialization?

These issues were resolved as follows:

- **Ordering of the fields**

By a vote of 8 to 1, people preferred the ordering envelope.payload.signature over the ordering signature.envelope.payload. Two reasons were cited: First, this allows for stream-mode operations, where consumers can begin operations based upon the contents of the envelope before the signature has arrived without having to buffer the signature, and where producers can compute the signature in parallel with the transmission of the envelope and payload. The counter-argument advanced by Paul Tarjan of Facebook (in abstentia) is that all languages have a string operation to split a string on the first occurrence of a character.

- **Ordering of the signing and encryption operations**

How to compose these operations depends upon scenario requirements. Goals identified include Integrity, Confidentiality + Integrity, and Non-Repudiation. Brad Hill identified four sets of relevant operations, which were public key signature, symmetric key MAC, symmetric key confidentiality + MAC, and key wrap/key transport/agreement with Diffie-Hellman. Some took the position that we should define a small set of fixed configurations that are known to safely achieve the intended goals; others argued for general composability of operations. This was the one topic that we had to defer to a follow-on session to be held the next day, due to time limitations.

- **What can be in an envelope (a small fixed set of things or is it extensible)?**

We reached a consensus that the envelope needs to be extensible (but should be extended only with great care).

- **What to sign (envelope.payload or just payload)?**

Given that the envelope is extensible and therefore may contain security-sensitive information, we reached a consensus (with input from Ben Laurie via IM) that the combination envelope.payload must be signed.

- **What can be in the payload (only sets of JSON token claims or arbitrary base64url encoded byte streams)?**

By a vote of 9 to 2, the group decided that the spec should support signing/encrypting of arbitrary base64url encoded byte streams. They also decided that the spec should define the syntax and semantics of a set of claims when what is being signed is a set of JSON claims.

- **Do we need to support multiple signatures?**

The group voted 5 to 2 that it must be possible to support multiple signatures in some manner. Two variants of multiple signatures were discussed: the “gateway case”, where additional signatures are added to a token as it is passed between parties, and the parallel case, where multiple parties sign the same contents up front. However the group also decided that it would be overly complicated to support multiple signatures in the compact

serialization. Support for multiple signatures was pushed to the JSON serialization (per the next issue).

- **Should we specify a JSON serialization as well as a compact serialization?**

The group decided by a vote of 11 to 1 that there were use cases for a JSON serialization, and that multiple signatures would be possible in that serialization. The syntax agreed upon simply uses the three base64url encoded fields, while allowing there to be parallel arrays of envelopes and signatures. Specifically, the syntax agreed upon was:

```
{"envelope":["<envelope 1 contents>",…,"<envelope N contents>"],
 "payload":"<payload contents>"]
 "signature":["<signature 1 contents>",…,"<signature N contents>"]
}
```

and where the i^{th} signature is computed on the concatenation of <envelope *i* contents>.<payload contents>.

« JSON Token Spec Results at IIW on Tuesday
JSON Token Naming Spec Results at IIW on Wednesday »

November 8, 2010

JSON Token Encryption Spec Results at IIW on Wednesday

We held a session on encryption for JSON Web Tokens at IIW on Wednesday, building upon the results from the JSON Tokens and No Base String sessions on Tuesday. Once again, substantial consensus emerged, which is described in the notes below.

These consensus decisions were in place by the start of the session:

- Some use cases for JSON tokens require encryption
- (plus all the decisions from the sessions on Tuesday)

It was agreed that these sets of high-level goals need to be achievable by application of signing and/or encryption:

- Integrity
- Confidentiality + Integrity
- Non-Repudiation (which also implies Integrity)
- Non-Repudiation + Confidentiality

Open issues identified at the start of the session were:

- Should encryption and signing be accomplished via (1) separate but composable encryption and signing operations, (2) use of a small set of recommended composite operations that achieve the high-level goals, or (3) allowing for both possibilities?
- Data format and how data formats affect streaming operations
- Order of signing and encryption operations
- Compress before encrypt?
- What are we encrypting (payload or payload + signature)?

The primary consensus in the room was to invent as little as possible by reusing work that other experts have done in the space, while adapting their work to a JSON context. Participants provided the following references to work to borrow from:

- Cryptographic Message Syntax (CMS): http://tools.ietf.org/html/rfc5652
- Table of Algorithm Suites from WS-SecurityPolicy 1.2, section 6.1: http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/ws-securitypolicy-1.2-spec-os.html#_Toc161826547
- XML Signature Syntax and Processing (Second Edition): http://www.w3.org/TR/xmldsig-core/
- XML Encryption Syntax and Processing: http://www.w3.org/TR/xmlenc-core/

- Defective Sign and Encrypt analysis: http://world.std.com/~dtd/sign_encrypt/sign_encrypt7.PDF
- The TLS Protocol Version 1.0: http://tools.ietf.org/html/rfc2246
- Transport Layer Security Protocol Compression Methods: http://tools.ietf.org/html/rfc3749
- Compressed Data Content Type for Cryptographic Message Syntax (CMS): http://tools.ietf.org/html/rfc3274
- DEFLATE Compressed Data Format Specification version 1.3: http://tools.ietf.org/html/rfc1951

Specific issues were resolved as follows:

- **Should encryption and signing be accomplished via (1) separate but composable encryption and signing operations, (2) use of a small set of recommended composite operations that achieve the high-level goals, or (3) allowing for both possibilities?**

The consensus was for (3) – that we should specify a small set of composite operations that will meet the needs of common use cases, while also enabling applications to compose encryption and signing operations in a general fashion, should the composite operations prove insufficient for their use cases. A subset of the composite algorithm suites in WS-SecurityPolicy 1.2 was suggested as an appropriate starting point. Paul Tarjan of Facebook had also suggested during the OpenID Summit on Monday that descriptive composite algorithm values be used, such as "AES-256-CBC HMAC-SHA-256".

- **Data format and how data formats affect streaming operations**

For the same reasons as discussed during the signing session, the group reaffirmed that the order of the fields should be envelope.payload.signature, with the envelope containing sufficient information to determine the nature of the contents of the remaining fields. This order enables streaming operations, where content is created or analyzed in parallel with its transmission or reception, to the maximum extent possible, and also potentially minimizes buffering requirements imposed upon implementations.

- **Order of signing and encryption operations**

It was recognized that no one-size-fits-all solution applies here and that different sets of operations are needed for different use cases. For instance, if non-repudiation is required, a signature of the plan text using public key cryptography must be present, which therefore must precede any other operations. Again, the group reaffirmed that we should reuse other work in this area to the extent possible.

- **Compress before encrypt?**

Several participants pointed out existing practice in this area, including the use of the DEFLATE compression algorithm prior to encryption by TLS and CMS. It was agreed that we should similarly document how to optionally perform compression before encryption for those use cases where it makes sense.

- **What are we encrypting (payload or payload + signature)?**

This was another area were the participants felt that we should reuse existing practice that has already been vetted by experts.

Special thanks go to Breno de Medeiros, whose crypto expertise was invaluable during this session, as well as Brad Hill, Diana Smetters and several other Googlers, John Bradley, Nat Sakimura, Joseph Holsten, Thomas Hardjono, Terry Hayes, Dick Hardt, Tony Nadalin, and others who contributed to this productive session.

« JSON Token Encryption Spec Results at IIW on Wednesday
JSON Public Key Spec Results at IIW on Thursday »

November 9, 2010

JSON Token Naming Spec Results at IIW on Wednesday

We held a session on naming for JSON Web Tokens (JWTs) at IIW, building upon the results from the JSON Tokens and No Base String sessions on Tuesday and the JSON Token Encryption session on Wednesday. Like the previous sessions, there was a clear consensus for the decisions the group made.

Names are needed for these specification elements:

1. Envelope parameters (such as the name of the signature parameter)
2. Claim names (such as the name of the issuer claim)
3. Algorithm names (such as the names representing the HMAC SHA-256 and AES-256-CBC algorithms)

The first issue tacked by the participants was whether short names should be used in order to keep tokens concise (and in particular, in order to have them be potentially usable in query strings for mobile phone browsers with 512-character URL limits), or whether to use longer, descriptive names. For instance, the name of an algorithm parameter could be either “alg” or “algorithm”. By a 7-2 vote, the participants opted for short names.

We next used the names in the current JWT spec to drive discussion on specific names in each category. In keeping with the decision to employ short names, Nat Sakimura suggested that the few names over three characters in length – specifically “keyid” and “curi” also be shortened to three-character names. The other participants concurred with this suggestion.

A discussion was held on behalf of Paul Tarjan of Facebook on defining a standard time-issued-at claim (which together with the expires claim, bounds the token lifetime). There was consensus that this claim should be defined by the specification.

George Fletcher led a discussion on whether an issued-to claim distinct from the audience claim should be defined. The group didn’t feel strongly about this, but voted 3-1 against including it. The participants noted that any claims meaningful to both parties can be defined as needed, so all claims need not be pre-defined in the specification.

The group discussed what algorithm names should be used. It was agreed that while each software package uses specific names for algorithms, because they tend to differ by software package, there is no compelling reason to choose one set over another. And indeed, given the group’s over-arching decision to use short names, people felt that employing short names such as “HS256” imposed no more burden on implementers that using longer names like “HMAC-SHA-256” or “http://www.w3.org/2001/04/xmldsig-

more#hmac-sha256". Thus, the short names in the current JWT spec were endorsed, with the understanding that additional names will be needed for encryption algorithm names and names of recommended algorithm combinations.

To help implementers, the group suggested that the specification include a table cross-referencing the algorithm name strings used in standard software packages and specifications. Breno de Medeiros supplied a link to the JCE algorithm names for inclusion in this table.

While not strictly on the topic of naming, the group held a discussion of how to factor the JWT specification or specifications so as to maximize its acceptance and adoption. The choices discussed were (1) a single specification, (2) a part one specification covering signing and claims and a part two specification covering encryption, and (3) three related specifications – one for signing, one for claims, and one for encryption. The consensus was for (2), since the normal use case will always include signed sets of claims, whereas people should only need to pay the price to understand encryption if they actually need to employ it.

Finally, Nat Sakimura asked if we wanted the branding of the specification or specifications to be more general than JSON Web Token (JWT), since the scope of the work is actually broader – encompassing JSON encodings for claims, signing, and encryption. Mike Jones took the position that, given that this is a composite spec including JWT claims, that we're better off branding it in a way that matches the common use case, and therefore and keeping the name JWT as the overall brand. The participants concurred.

« JSON Token Naming Spec Results at IIW on Wednesday
AD FS 2.0 Interop Step-By-Step Guide: Ping Identity PingFederate »

November 9, 2010

JSON Public Key Spec Results at IIW on Thursday

The final session at IIW related to JSON Web Tokens (JWTs) explored whether and how to represent public key information as JWTs or other JSON structures as an alternative to X.509 certificates. Thanks to Breno de Medeiros for taking notes, which I've pasted in below:

Issue/Topic: Public Key Certificates as JWT
Session: Thursday 1E
Convener: Mike Jones, Microsoft
Notes-taker(s): Breno de Medeiros
Tags: If and how to represent public key certificates as JSON Web Tokens
Discussion notes:

- Certificate installation a difficult and core technical obstacle in configuring security
- Not all cases require PKI validation; motivation examples given by J. Panzer et. al., drove the proposal for the Magic Signatures specs
- In the absence of PKI certificates, it's not possible to 'preserve' the security context around fetching the certificate
- Is there a need to invent another type of JSON-based certificate? Do we have a need for certificates in addition to bare keys?
- Why re-invent X.509? Create a JSON binding for the subset of KeyInfo from X.509 that is needed to advertise keys
- After reviewing the KeyInfo, decided that the part of it of interest is trivially small and already described in competing proposals
- Even a JWT is too complex, only need to create a simple descriptor for the key in JSON
- Key_id needed

Decision: Go with simple approach

- Keep this mini-spec separate from JWT and cross-reference? Or include this in the expanded spec of JWT to include encryption?

Decision: Keep specs separate

- Need to allow this to have a URL-safe representation such as compact JWT?

Examples of what these representations might look like are as follows:

```
{"keyvalues":
  [
    {"alg":"ECDSA",
     "x":"MKBCTNIcKUSDii11ySs3526iDZ8AiTo7Tu6KPAqv7D4",
     "y":"4Etl6SRW2YiLUrN5vfvVHuhp7x8PxltmWWlbbM4IFyM",
     "keyid":"1"},

  {"alg":"RSA",
  "modulus":
"0vx7agoebGcQSuuPiLJXZptN9nndrQmbXEps2aiAFbWhM78LhWx4cbbfAAtVT86zwu1RK7a
PFFxuhDR1L6tSoc_BJECPebWKRXjBZCiFV4n3oknjhMstn64tZ_2W-
5JsGY4Hc5n9yBXArwl93lqt7_RN5w6Cf0h4QyQ5v-
65YGjQR0_FDW2QvzqY368QQMicAtaSqzs8KJZgnYb9c7d0zgdAZHzu6qMQvRL5hajrn1n91C
bOpbISD08qNLyrdkt-bFTWhAI4vMQFh6WeZu0fM4lFd2NcRwr3XPksINHaQ-
G_xBniIqbw0Ls1jF44-csFCur-kEgU8awapJzKnqDKgw",
  "exponent":"AQAB",
  "keyid":"2"}
 ]
}
```

Near the end of the discussion, it was pointed out that want we are proposing is much closer to the XMLDSIG KeyValue element than the KeyInfo element.

The participants recognize that the security of these raw keys is dependent upon the security of the mechanisms for distributing them – in most cases TLS.

References:

- XML Signature Syntax and Processing (Second Edition): http://www.w3.org/TR/xmldsig-core/
- Using the Elliptic Curve Signature Algorithm (ECDSA) for XML Digital Signatures: http://tools.ietf.org/html/rfc4050
- Additional XML Security Uniform Resource Identifiers (URIs): http://tools.ietf.org/html/rfc4051
- Magic Signatures: http://salmon-protocol.googlecode.com/svn/trunk/draft-panzer-magicsig-experimental-00.html