U-Prove Extensions

Christian Paquin, Microsoft Research

August 4th 2014

Introduction

The U-Prove Cryptographic Specification[1] focuses on the core U-Prove capabilities; the specified features were selected to simplify implementation and integration into existing systems, while meeting the needs of a wide array of scenarios. By design, the specification provides extension points, making it possible to extend the core capabilities to meet additional needs.

This paper describes recently released features compatible with the U-Prove technology. The reader is assumed to be familiar with the technology, and is referred to the technology overview for an introduction.[2]

Extensions

The U-Prove Cryptographic Specification defines the core set of features, the main ones being token issuance and presentation unlinkability, and subset disclosure of the attributes. Additional features are however possible and desired to increase the usefulness of the technology. The additional features are provided as protocol extensions, each one being explained in the following sections.

Note that two previously-released extensions, accumulator-based revocation and ID escrow through verifiable encryption, have already been released and explained in the "Privacy and accountability in identity systems: the best of both worlds" white paper;[3] these features will not be covered in this paper.

The extension features have been implemented in a C# SDK;[4] samples are also available to demonstrate them.

Collaborative issuance

Issuers typically know the attributes they encode in U-Prove tokens; this allows the verifiers to trust that the attributes were properly vetted by their emitters. In some cases, however, it makes sense for an issuer to issue tokens with attributes it doesn't know. We will explore two scenarios enabled by the collaborative issuance extension.

[1] http://research.microsoft.com/apps/pubs/default.aspx?id=166969
[2] http://research.microsoft.com/apps/pubs/default.aspx?id=166980
[3] http://research.microsoft.com/apps/pubs/?id=200815
[4] http://research.microsoft.com/en-us/downloads/edd75bcb-371c-4636-abcb-a431f43ecf5f/

To learn more, consult the collaborative issuance extension specification.[5]

Trusted attribute providers

In some scenarios, the token attributes are provided to the issuer by some trusted attribute providers. For example, imagine an issuance service similar to a public certificate authority, who issues tokens on behalf of different organizations (its customers). In turns, users of these organizations obtain tokens from that issuer. However, for security, privacy, and liability reasons, the issuer is not interested in learning the attribute values of its customers' users. Therefore, the attribute providers and the issuer can collaborate to issue tokens without the issuer having to see the attribute values. To achieve this, the U-Prove issuance protocol is slightly modified to allow an attribute provider to provide the attributes to the issuer in an opaque manner as input to the protocol. The steps are illustrated in Figure 1:

1) After authenticating the user, the attribute provider encodes the token attributes into a "blinded" opaque cryptographic value, and sends the attribute values to the user along with the unblinding information.[6]
2) The attribute provider then sends the blinded cryptographic value to the issuer over a trusted channel.
3) Finally, the user and the issuer perform the issuance protocol resulting in one or more U-Prove tokens. The user un-blinds the tokens using the value received from the attribute provider in step 1.

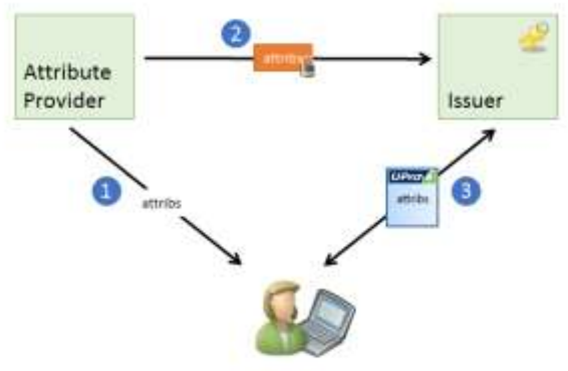

Figure 1: trusted attribute provider

In this scenario, it is understood that the issuer never sees and therefore does not vet the attribute values; it merely manages the issuance private key, and could offer further security protections (for example, revocation and ID escrow services) shared by many attribute providers. There is nothing in the U-Prove token that indicates that collaborative issuance was used, and if so, which attribute provider was involved. An application may want to encode this data explicitly in the token (for example, by adding some attribute provider information in the token information field).

---

[5] http://research.microsoft.com/apps/pubs/?id=219670

[6] The opaque cryptographic value is not encrypted or hashed; it is a blinded version of a value that the issuer would have calculated would it have received the attribute values in clear. The unblinding information is also sent to the user with the attributes, allowing her to un-blind the issued token's public key. This blinding prevents the issuer from guessing the values encoded in the token; even if the token contained a single Boolean attribute (for example, "over-21?"), then it would be infeasible for the issuer to figure out if the attribute value is true or false.

Carried-over attribute

In some cases, in might be needed to encode a user-provided attribute into a token, without the issuer learning its value. One scenario would be to carry-over an attribute from an existing token into a new one. Assume, for example, that an attribute encodes a unique user identifier used for revocation purposes. Since issuers might want to share the revocation infrastructure, it makes sense to have all of a user's tokens encode the same value. Disclosing this value at issuance might, however, reveal too much information about the user. This feature allows a user to carry-over the attribute into a new token without disclosing its value to the issuer; the issuer would be convinced that the value came from another token it just validated. The steps are illustrated in Figure 2:

1) The user obtains a U-Prove token from the 1st issuer.
2) Later, the user visits the 2nd issuer, wishing to carry-over an attribute from the first token. She presents her token to the 2nd issuer, hiding the attribute values.[7]
3) The user and the 2nd issuer perform the issuance protocol, encoding the attributes that are carried-over from the first token. Additional attributes can be added by the issuer as well.

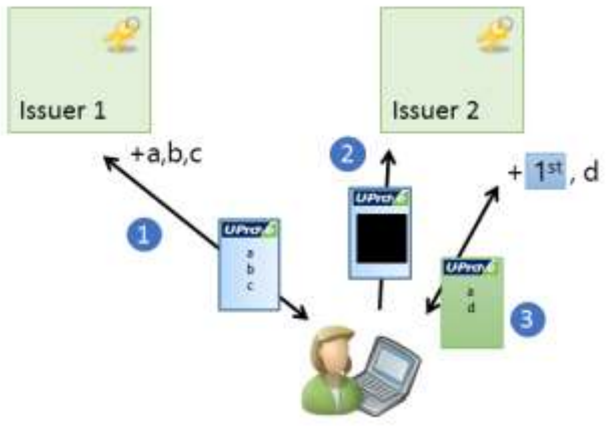

Figure 2: carry-over of attributes

The carrying-over of attributes is demonstrated in the `CollaborativeIssuanceAndEqualitySample` SDK sample.

Attribute equality proof

To prove that an attribute value is equal to a certain value, the user simply has to disclose it in a presentation proof. It is however possible to prove that a value is equal to a committed value or to another token attribute value unknown to the verifier without disclosing it, as illustrated in Figure 3. Imagine that two tokens share one attribute value (step 1), say a unique user identifier; then the equality proof extension allows the user to prove the two attributes are equal without the verifier learning anything else about the value (step 2). This is useful to convince the verifier that attributes

[7] The user could present other attributes to the issuer. In general, using any other extension, the user could present any property about the carried-over and/or other attributes, to convince the issuer of the token meets its application-specific issuance policy. The user must commit in the presentation protocol to the attribute values she wishes to carry-over to the new token.

pooled together from different tokens belong to the same user (and were not "borrowed" from someone else).

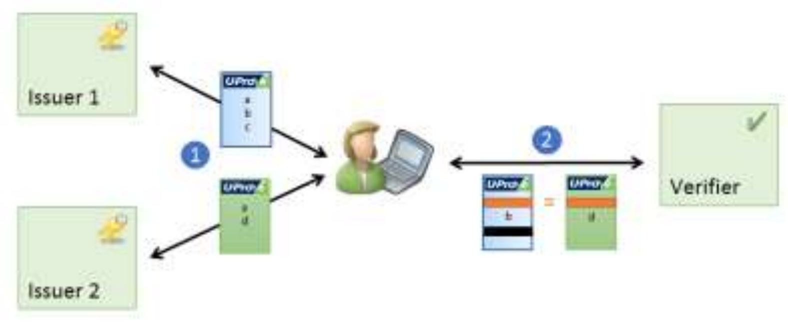

Figure 3: token attribute equality

Combined with the carrying-over of attributes feature described above, this allows issuers to extend an existing token by issuing a new token bound to the first one, carrying over a unique user identifier and having the user prove that the identifiers are the same in each token. This is demonstrated in the `CollaborativeIssuanceAndEqualitySample` SDK sample. To learn more, consult the equality proof extension specification.[8]

Attribute inequality proof

Proving that an attribute is not equal to a target value is also possible, using the inequality proof extension. The target value is either known to the verifier, or not. If the target value is known, then this feature can be used as a revocation mechanism, allowing a user to prove that her identifier does not appear on a revocation list, as illustrated in Figure 4.

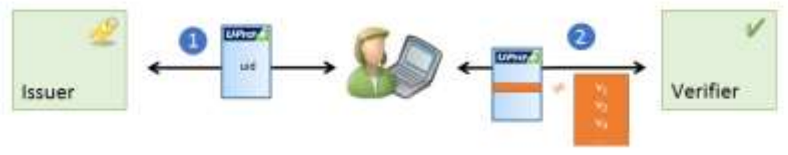

Figure 4: inequality proof used for revocation

Alternatively, the user can prove that two token attributes are different, without revealing them, as illustrated in Figure 5.

[8] http://research.microsoft.com/apps/pubs/?id=219672

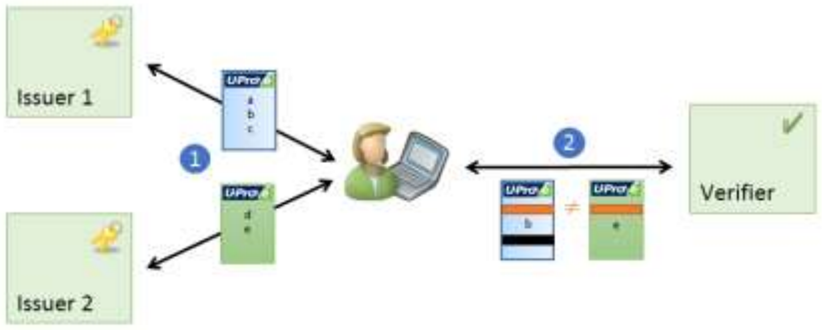

Figure 5: token attribute inequality

To learn more, consult the inequality proof extension specification.[9]

Set membership proof

Another useful feature is to prove that a token attribute value is within a set, without disclosing which set value it is, as illustrated in Figure 6. The set values are known to the verifier, but it cannot determine which value is encoded in the presented token.

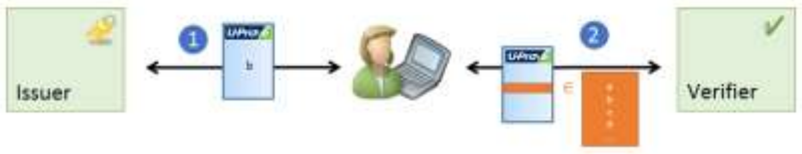

Figure 6: set membership proof

The size of the resulting proof grows linearly with the size of the set;[10] this must be taken into consideration when designing a system using this feature. Set membership is demonstrated in the `SetMembershipSample` SDK sample. To learn more, consult the set membership proof extension specification.[11]

Range proof

Proving that an attribute value is within a range is one of the most interesting features of minimal disclosure technologies. Indeed, being able to prove that, for example, you are over-21 without disclosing your date of birth sounds infeasible, but it is made possible using the range proof extension.

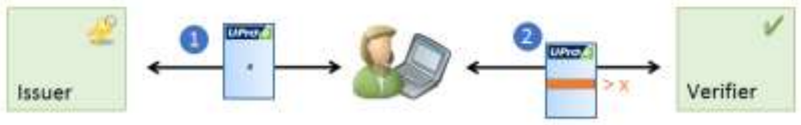

Figure 7: range proof

[9] http://research.microsoft.com/apps/pubs/?id=219673

[10] Technically, one proof is created for each set element. Only one of the proof will be valid, but the verifier won't know which one, and will be convinced that the token attribute value is equal to one element in the set.

[11] http://research.microsoft.com/apps/pubs/?id=219675

Numerical attributes must be encoded directly[12] into the token. The extension supports proving that a value is “less than”, “less than or equal to”, “greater than”, or “greater than or equal to” a target value, known or unknown (for example, coming from another token) to the verifier. The extension provides helper methods to compare age and date attributes.

The range proof protocol functions by comparing the bit decomposition of the compared values; therefore, a significant number of subproofs are created during the protocol. It is advised to minimize the ranges to be used in these proofs. For small enough ranges, a set membership proof might be more efficient. If performance is a critical issue in an application, multiple attributes might alternatively be encoded in the token to answer common queries from verifiers.[13]

Range proofs are demonstrated in the `RangeProofSample` SDK sample. To learn more, consult the range proof extension specification.[14]

Acknowledgments

This paper highlights the work of my esteemed MSR colleagues Greg Zaverucha, Lan Nguyen, and Melissa Chase, and Mira Belenkiy.

Resources

You can learn more about U-Prove by visiting http://www.microsoft.com/u-prove. A C# SDK implementing the extensions presented in this paper can be found at http://bit.ly/uproveextensions.

Microsoft

[12] Rather than being hashed. In other words, the Issuer Parameters E value must be set to 0 for the attribute being used in a range proof.
[13] For example, instead of using a range proof on a date of birth to prove that the user is of age, the issuer could encode various Boolean values (for example: over-13, over-16, over-18, over-19, over-21, and over-65) which could answer the “of age” question in most jurisdictions around the world.
[14] http://research.microsoft.com/apps/pubs/?id=219674