Internet Engineering Task Force
Internet-Draft
Intended status: Informational
Expires: October 28, 2017

F. Hao, Ed.
Newcastle University (UK)

April 26, 2017

Schnorr NIZK Proof: Non-interactive Zero Knowledge Proof for Discrete Logarithm
draft-hao-schnorr-06

Abstract

This document describes Schnorr NIZK proof, a non-interactive variant of the three-pass Schnorr identification scheme. The Schnorr NIZK proof allows one to prove the knowledge of a discrete logarithm without leaking any information about its value. It can serve as a useful building block for many cryptographic protocols to ensure the participants follow the protocol specification honestly. This document specifies the Schnorr NIZK proof in both the finite field and the elliptic curve settings.

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 October 28, 2017.

Copyright Notice

Table of Contents

1. Introduction

A well-known principle for designing robust public key protocols states as follows: "Do not assume that a message you receive has a particular form (such as g^r for known r) unless you can check this" [AN95]. This is the sixth of the eight principles defined by Ross Anderson and Roger Needham at Crypto'95. Hence, it is also known as the "sixth principle". In the past thirty years, many public key protocols failed to prevent attacks, which can be explained by the violation of this principle [Hao10].

While there may be several ways to satisfy the sixth principle, this

document describes one technique that allows one to prove the knowledge of a discrete logarithm (e.g., r for g^r) without revealing its value. This technique is called the Schnorr NIZK proof, which is a non-interactive variant of the three-pass Schnorr identification scheme [Stinson06]. The original Schnorr identification scheme is

made non-interactive through a Fiat-Shamir transformation [FS86], assuming that there exists a secure cryptographic hash function (i.e., the so-called random oracle model).

The Schnorr NIZK proof can be implemented over a finite field or an elliptic curve (EC). The technical specification is basically the same, except that the underlying cyclic group is different. For completeness, this document describes the Schnorr NIZK proof in both the finite field and the EC settings.

1.1. Requirements Language

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].

1.2. Notations

The following notations are used in this document:

- Alice: the assumed identity of the prover in the protocol
- Bob: the assumed identity of the verifier in the protocol
- a | b: a divides b
- a || b: concatenation of a and b
- [a, b]: the interval of integers between and including a and b
- t: the bit length of the challenge chosen by Bob
- H: a secure cryptographic hash function
- p: a large prime

- q: a large prime divisor of p-1, i.e., q | p-1
- Zp*: a multiplicative group of integers modulo p
- Gq: a subgroup of Zp* with prime order q
- g: a generator of Gq
- g^x: g raised to the power of x
- a mod b: a modulo b

- Fp: a finite field of p elements where p is a prime
- E(Fp): an elliptic curve defined over Fp
- G: a generator of the subgroup over E(Fp) with prime order n
- n: the order of G
- h: the cofactor of the subgroup generated by G, which is equal to the order of the elliptic curve divided by n
- P x [b]: multiplication of a point P with a scalar b over E(Fp)

2. Schnorr NIZK Proof over Finite Field

2.1. Group Parameters

When implemented over a finite field, the Schnorr NIZK proof may use the same group setting as DSA [FIPS186-4]. Let p and q be two large primes with q | p-1. Let Gq denote the subgroup of Zp* of prime order q, and g be a generator for the subgroup. Refer to NIST [1] for values of (p, q, g) that provide different security levels. A level of 128-bit security or above is recommended. Here DSA groups are used only as an example. Other multiplicative groups where the discrete logarithm problem (DLP) is intractable are also suitable for the implementation of the Schnorr NIZK proof.

2.2. Schnorr Identification Scheme

The Schnorr identification scheme runs interactively between Alice (prover) and Bob (verifier). In the setup of the scheme, Alice publishes her public key X = g^x mod p where x is the private key chosen uniformly at random from [0, q-1].

The protocol works in three passes:

1. Alice chooses a number v uniformly at random from [0, q-1] and computes V = g^v mod p. She sends V to Bob.

2. Bob chooses a challenge c uniformly at random from [0, 2^t-1], where t is the bit length of the challenge (say t = 160). Bob sends c to Alice.

3. Alice computes b = v - x * c mod q and sends it to Bob.

At the end of the protocol, Bob performs the following checks. If any check fails, the identification is unsuccessful.

1. To verify X is within [1, p-1] and X^q = 1 mod p;

2. To verify V = g^r * X^c mod p.

The first check ensures that X is a valid public key, hence the discrete logarithm of X with respect to the base g actually exists. It is worth noting that some applications may specifically exclude the identity element as a valid public key. In that case, one shall check X is within [2, p-1] instead of [1, p-1].

The process is summarized in the following diagram.

Information Flows in Schnorr Identification Scheme over Finite Field

```
       Alice                                  Bob
      -------                                -----

choose random v from [0, q-1]

compute V = g^v mod p     -- V ->

compute b = v-x*c mod q  <- c -- choose random c from [0, 2^t-1]
```

```
                          -- b -> check 1) X is a valid public key
                                        2) V = g^b * X^c mod p
```

2.3. Non-Interactive Zero-Knowledge Proof

The Schnorr NIZK proof is obtained from the interactive Schnorr identification scheme through a Fiat-Shamir transformation [FS86]. This transformation involves using a secure cryptographic hash function to issue the challenge instead. More specifically, the challenge is redefined as c = H(g || g^v || g^x || UserID || OtherInfo), where UserID is a unique identifier for the prover and OtherInfo is optional data. Here, the hash function H shall be a secure cryptographic hash function, e.g., SHA-256, SHA-384, SHA-512, SHA3-256, SHA3-384 and SHA3-512. The bit length of the hash output should be at least equal to the order q of the considered subgroup.

The OtherInfo is defined to allow flexible inclusion of contextual information (also known as "labels" in [ABM15]) in the Schnorr NIZK proof so that the technique defined in this document can be generally useful. For example, some security protocols built on top of the Schnorr NIZK proof may wish to include more contextual information such as the protocol name, timestamp and so on. The exact items (if any) in OtherInfo shall be left to specific protocols to define. However, the format of OtherInfo in any specific protocol must be fixed and explicitly defined in the protocol specification.

Within the hash function, there must be a clear boundary between any two concatenated items. It is recommended that one should always prepend each item with a 4-byte integer that represents the byte length of that item. The OtherInfo may contain multiple sub-items. In that case, the same rule shall apply to ensure a clear boundary between adjacent sub-items.

2.4. Computation Cost

In summary, to prove the knowledge of the exponent for X = g^x, Alice generates a Schnorr NIZK proof that contains: {UserID, OtherInfo, V = g^v mod p, r = v - x*c mod q}, where c = H(g || g^v || g^x || UserID || OtherInfo).

To generate a Schnorr NIZK proof, the cost is roughly one modular

exponentiation: that is to compute g^v mod p. In practice, this exponentiation may be pre-computed in the off-line manner to optimize efficiency. The cost of the remaining operations (random number generation, modular multiplication and hashing) is negligible as compared with the modular exponentiation.

To verify the Schnorr NIZK proof, the cost is approximately two exponentiations: one for computing X^q mod p and the other for computing g^r * X^c mod p. (It takes roughly one exponentiation to compute the latter using a simultaneous exponentiation technique as described in [MOV96].)

3. Schnorr NIZK Proof over Elliptic Curve

3.1. Group Parameters

When implemented over an elliptic curve, the Schnorr NIZK proof may use the same EC setting as ECDSA [FIPS186-4]. For the illustration purpose, only curves over the prime fields (e.g., NIST P-256) are described here. Other curves over the binary fields (see [FIPS186-4]) that are suitable for ECDSA can also be used for implementing the Schnorr NIZK proof. Let E(Fp) be an elliptic curve defined over a finite field Fp where p is a large prime. Let G be a base point on the curve that serves as a generator for the subgroup over E(Fp) of prime order n. The cofactor of the subgroup is denoted h, which is usually a small value (not more than 4). Details on EC operations, such as addition, negation and scalar multiplications, can be found in [MOV96]. Date types and conversions including elliptic-curve-point-to-octet-string and vice versa can be found in Section 2.3 of [SEC1]. Here the NIST curves are used only as an example. Other secure curves such as Curve25519 are also suitable for the implementation as long as the elliptic curve discrete logarithm problem (ECDLP) remains intractable.

3.2. Schnorr Identification Scheme

In the setup of the scheme, Alice publishes her public key Q = G x [x] where x is the private key chosen uniformly at random from [1, n-1].

The protocol works in three passes:

1. Alice chooses a number v uniformly at random from [1, n-1] and computes V = G x [v]. She sends V to Bob.

2. Bob chooses a challenge c uniformly at random from [0, 2^t-1], where t is the bit length of the challenge (say t = 80). Bob sends c to Alice.

3. Alice computes b = v - x * c mod n and sends it to Bob.

At the end of the protocol, Bob performs the following checks. If any check fails, the verification is unsuccessful.

1. To verify X is a valid point on the curve and X x [h] is not the point at infinity;

2. To verify V = G x [b] + Q x [c].

The first check ensures that X is a valid public key, hence the discrete logarithm of X with respect to the base G actually exists. Unlike in the DSA-like group setting where a full modular exponentiation is required to validate a public key, in the ECDSA-like setting, the public key validation incurs almost negligible cost due to the cofactor being small (e.g., 1, 2 or 4).

The process is summarized in the following diagram.

Information Flows in Schnorr Identification Scheme over Elliptic Curve

```
Alice                                  Bob
-------                                -----

choose random v from [1, n-1]

compute V = G x [v]           -- V ->

compute b = v - x * c mod n   <- c -- choose random c from [0, 2^t-1]

                              -- b -> check 1) X is a valid public key
                                           2) V = G x [b] + Q x [c]
```


3.3. Non-Interactive Zero-Knowledge Proof

Same as before, the non-interactive variant is obtained through a Fiat-Shamir transformation [FS86], by using a secure cryptographic hash function to issue the challenge instead. The challenge c is defined as c = H(G || V || Q || UserID || OtherInfo), where UserID is a unique identifier for the prover and OtherInfo is optional data as explained earlier.

3.4. Computation Cost

In summary, to prove the knowledge of the discrete logarithm for Q = G x [x] with respect to base G over the elliptic curve, Alice generates a Schnorr NIZK proof that contains: {UserID, OtherInfo, V = G x [v], r = v - x*c mod n}, where c = H(G || V || Q || UserID || OtherInfo).

To generate a Schnorr NIZK proof, the cost is one scalar multiplication: that is to compute G x [v].

To verify the Schnorr NIZK proof in the EC setting, the cost is approximately one multiplication over the elliptic curve: i.e., computing G x [r] + Q x [c] (using the same simultaneous computation technique as before). The cost of public key validation in the EC setting is essentially free.

4. Variants of Schnorr NIZK proof

In the finite field setting, the prover sends (V, b) (along with userID and OtherInfo), and the verifier first computes c, and then checks for V = g^b * X^c mod p. This requires the transmission of an element V of Zp, whose size is typically between 2048 and 3072 bits, and an element b of Zq whose size is typically between 224 and 256 bits. It is possible to reduce the amount of transmitted data to two elements of Zq as below.

In the modified variant, the prover works exactly the same as before, except that it sends (c, b) instead of (V, b). The verifier computes V = g^b * X^c mod p and then checks whether H(g || V || g^x || UserID || OtherInfo) = c. The security of this modified variant follows from the fact that one can compute V from (c, b) and c from (V, b). Therefore, sending (c, b) is equivalent to sending (V, c, b), which in turn is equivalent to sending (V, b). Thus, the size of the Schnorr NIZK proof is significantly reduced. However, the computation costs for both the prover and the verifier stay the same.

The same optimization technique also applies to the elliptic curve setting by replacing (V, b) with (c, b), but the benefit is much

limited. When V is encoded in the compressed form, this optimization only saves 1 bit. The computation costs for generating and verifying the NIZK proof remain the same as before.

5. Applications of Schnorr NIZK proof

Some key exchange protocols, such as J-PAKE [HR08] and YAK [Hao10], rely on the Schnorr NIZK proof to ensure participants have the knowledge of discrete logarithms, hence following the protocol specification honestly. The technique described in this document can be directly applied to those protocols.

The inclusion of OtherInfo also makes the Schnorr NIZK proof generally useful and flexible to cater for a wide range of applications. For example, the described technique may be used to allow a user to demonstrate the Proof-Of-Possession (PoP) of a long-term private key to a Certificate Authority (CA) during the public key registration phrase. It must be ensured that the hash contains data that links the proof to one particular key registration procedure, e.g., by including the CA name, the expiry date, the applicant's email contact and so on to the OtherInfo. In this case, the Schnorr NIZK proof is functionally equivalent to a self-signed Certificate Signing Request generated by using DSA or ECDSA.

6. Security Considerations

The Schnorr identification protocol has been proven to satisfy the following properties, assuming that the verifier is honest and the discrete logarithm problem is intractable (see [Stinson06]).

1. Completeness -- a prover who knows the discrete logarithm is always able to pass the verification challenge.

2. Soundness -- an adversary who does not know the discrete logarithm has only a negligible probability (i.e., 2^(-t)) to pass the verification challenge.

3. Honest verifier zero-knowledge -- a prover leaks no more than one bit information to the honest verifier: whether the prover knows the discrete logarithm.

The Fiat-Shamir transformation is a standard technique to transform a three-pass interactive Zero Knowledge Proof protocol (in which the verifier chooses a random challenge) to a non-interactive one, assuming that there exists a secure cryptographic hash function. Since the hash function is publicly defined, the prover is able to

compute the challenge by itself, hence making the protocol non-interactive. In this case, the hash function (more precisely, the

random oracle in the security proof) implements an honest verifier, because it assigns a uniformly random challenge c to each commitment (g^v or G x [v]) sent by the prover. This is exactly what an honest verifier would do.

It is important to note that in Schnorr's identification scheme and its non-interactive variant, a secure random number generator is required. In particular, bad randomness in v may reveal the secret discrete logarithm. For example, suppose the same random value V = g^v mod p is used twice by the prover (e.g., because its random number generator failed), but the verifier chooses different challenges c and c' (or the hash function is used on two different OtherInfo data, producing two different values c and c'). The adversary now observes two proof transcripts (V, c, b) and (V, c', b'), based on which he can compute the secret key x by: (b-b')/(c'-c) = (v-x*c-v+x*c')/(c'-c) = x mod q.

More generally, such an attack may even work for a slightly better (but still bad) random number generator, where the value v is not repeated, but the adversary knows a relation between two values v and v' such as v' = v + w for some known value w. Suppose the adversary observes two proof transcripts (V, c, b) and (V', c', b'). He can compute the secret key x by: (b-b'+w)/(c'-c) = (v-x*c-v-w+x*c'+w)/(c'-c) = x mod q. This example reinforces the importance of using a secure random number generator to generate the ephemeral secret v in Schnorr's schemes.

Finally, when a security protocol relies on the Schnorr NIZK proof for proving the knowledge of a discrete logarithm in a non-interactive way, the threat of replay attacks shall be considered. For example, the Schnorr NIZK proof might be replayed back to the prover itself (to introduce some undesirable correlation between items in a cryptographic protocol). This particular attack is prevented by the inclusion of the unique UserID into the hash. The verifier shall check the prover's UserID is a valid identity and is different from its own. Depending on the context of specific protocols, other forms of replay attacks should be considered, and appropriate contextual information included into OtherInfo whenever necessary.

7. IANA Considerations

This document has no actions for IANA.

8. Acknowledgements

The editor of this document would like to thank Dylan Clarke, Robert Ransom, Siamak Shahandashti, Robert Cragie, Stanislav Smyshlyaev and Tibor Jager for many useful comments. Tibor Jager contributed the optimization technique and discussion the vulnerability issue when the ephemeral secret v is not generated randomly. This work is supported by the EPSRC First Grant (EP/J011541/1) and the ERC Starting Grant (No. 306994).

9. References

9.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[SEC1] "Standards for Efficient Cryptography. SEC 1: Elliptic Curve Cryptography", SECG SEC1-v2, May 2004, <http://www.secg.org/sec1-v2.pdf>.

[ABM15] Abdalla, M., Benhamouda, F., and P. MacKenzie, "Security of the J-PAKE Password-Authenticated Key Exchange Protocol", IEEE Symposium on Security and Privacy, May 2015.

[AN95] Anderson, R. and R. Needham, "Robustness principles for public key protocols", Proceedings of the 15th Annual International Cryptology Conference on Advances in Cryptology, 1995.

[FS86] Fiat, A. and A. Shamir, "How to Prove Yourself: Practical Solutions to Identification and Signature Problems", Proceedings of the 6th Annual International Cryptology Conference on Advances in Cryptology, 1986.

[MOV96] Menezes, A., Oorschot, P., and S. Vanstone, "Handbook of Applied Cryptography", 1996.

[Stinson06] Stinson, D., "Cryptography: Theory and Practice (3rd Edition)", CRC, 2006.

9.2. Informative References

[FIPS186-4] "Federal Information Processing Standards Publication 186-4: Specifications for the Digital Signature Standard (DSS)", July 2013, <http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf>.

[HR08] Hao, F. and P. Ryan, "Password Authenticated Key Exchange by Juggling", the 16th Workshop on Security Protocols, May 2008.

[Hao10] Hao, F., "On Robust Key Agreement Based on Public Key Authentication", the 14th International Conference on Financial Cryptography and Data Security, February 2010.

9.3. URIs

[1] http://csrc.nist.gov/groups/ST/toolkit/documents/Examples/DSA2_All.pdf

Author's Address

Feng Hao (editor)
Newcastle University (UK)

Claremont Tower, School of Computing Science, Newcastle University
Newcastle Upon Tyne
United Kingdom

Phone: +44 (0)191-208-6384
EMail: feng.hao@ncl.ac.uk