**Schnorr NIZK Proof: Non-interactive Zero Knowledge Proof for Discrete
Logarithm
draft-hao-schnorr-05**

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 May 18, 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 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:

o Alice: the assumed identity of the prover in the protocol

o Bob: the assumed identity of the verifier in the protocol

o a || b: concatenation of a and b

o t: the bit length of the challenge chosen by Bob

o H: a secure cryptographic hash function

o p: a large prime

o q: a large prime divisor of p-1, i.e., q | p-1

o Zp*: a multiplicative group of integers modulo p

o Gq: a subgroup of Zp* with prime order q

o g: a generator of Gq

o g^x: g raised to the power of x

o a mod b: a modulo b

o Fq: a finite field of q elements where q is a prime

o E(Fq): an elliptic curve defined over Fq

o G: a generator of the subgroup over E(Fq) with prime order n

o   n: the order of G

o   h: the cofactor of the subgroup generated by G, as defined by h
    = |E(Fq)|/n

o   P x [b]: multiplication of a point P with a scalar b over E(Fq)

o   P.x: the x coordinate of a point P over E(Fq)

[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. 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. 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 value X must be an
element in the subgroup Gq, which anyone can verify. This is to
ensure that the discrete logarithm of X with respect to the base g
actually exists.

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 = 80). 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 checks if the following equality
holds: V = g^b * X^c mod p. The verification succeeds only if the
equality holds. The process is summarized in the following diagram.

Information Flows in Schnorr Identification Scheme

```
         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 if 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
collision-resistant. Recommended hash functions include SHA-256,
SHA-384, SHA-512, SHA3-256, SHA-384 and SHA3-512.

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 the
concatenated items. Usually, the boundary is implicitly defined once
the length of each item is publicly known. However, in the general
case, it is safer to define the boundary explicitly. It is
recommended that one should always prepend each item with a 4-byte
integer that represents the byte length of the 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 following computations shall be
   performed.

   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

   Hence, the cost of verifying a Schnorr NIZK proof 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].)

   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]. Also note that in
   the DSA-like group setting, it requires a full modular exponentiation
   to validate a public key, but in the ECDSA-like setting, the public
   key validation incurs almost negligible cost due to the cofactor
   being very small (see [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, e.g., NIST P-256, P-384, and P-521
   [NISTCurve]. Let E(Fq) be an elliptic curve defined over a finite
   field Fq where q is a large prime. Let G be a base point on the
   curve that serves as a generator for the subgroup over E(Fq) 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]. 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
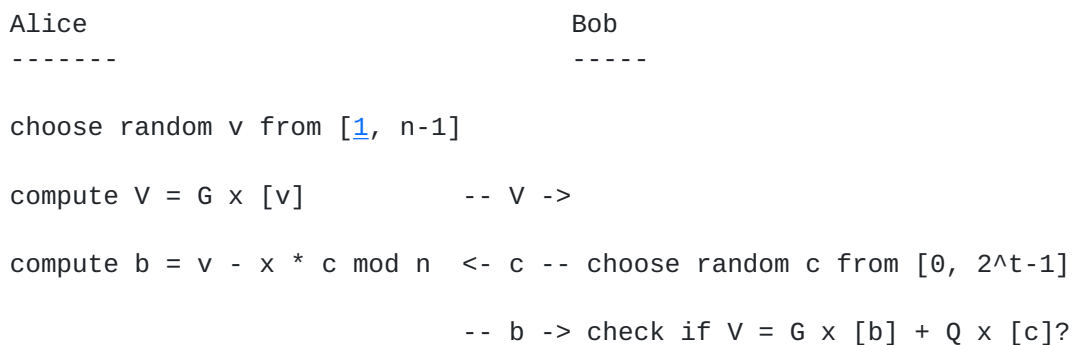problem (ECDLP) remains intractable.

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 value Q must be an element in the subgroup over the
elliptic curve, which anyone can verify.

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 checks if the following equality
holds: V = G x [b] + Q x [c]. The verification succeeds only if the
equality holds. The process is summarized in the following diagram.

Information Flows in Schnorr Identification Scheme

```
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 if 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. Note that G, V and Q
are points on the curve. In practice, it is sufficient to include
only the x coordinate of the point into the hash function. Hence,
let G.x, V.x and Q.x be the x coordinates of these points
respectively. The challenge c is defined as c = H(G.x || V.x ||

Q.x || 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.x || V.x || Q.x ||
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 following
computations shall be performed.

1. To verify Q is a valid public key in the subgroup over E(Fq)

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

In the EC setting where the cofactor is small (say 1, 2 or 4),
validating the public key Q is essentially free (see [MOV96]).   The
cost of verifying a Schnorr NIZK proof in the EC setting 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).

4. 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 in the protocol
follow the specification honestly. Hence, 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 sufficiently 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. Accordingly, the OtherInfo should include
extra information such as the CA name, the expiry date, the
applicant's email contact and so on. In this case, the Schnorr NIZK
proof is equivalent to a self-signed Certificate Signing Request
generated by using DSA or ECDSA, except that its security is
underpinned by well-established security proofs [Stinson06] while
equivalent proofs are lacking in DSA or ECDSA.

5. 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 (collision-resistant) 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. The assumption of an honest verifier naturally
   holds because the verifier can be anyone.

   A non-interactive Zero Knowledge Proof is often called a signature
   scheme. However, it should be noted that the Schnorr NIZK proof
   described in this document is different from the original Schnorr
   signature scheme (see [Stinson06]) in that it is specifically
   designed as a proof of knowledge of the discrete logarithm rather
   than a general-purpose digital signing algorithm.

   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.

[6](). IANA Considerations

This document has no actions for IANA.

[7](). Acknowledgements

[8](). References

[8.1](). 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]()>.

   [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.

[8.2](). Informative References

   [NISTCurve]
               "Recommended Elliptic Curves for Federal Government use",
               July 1999,
               <http://csrc.nist.gov/groups/ST/toolkit/documents/dss/
               NISTReCur.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.

8.3. 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