

Independent Submission
Request for Comments: 8235
Category: Informational
ISSN: 2070-1721

F. Hao, Ed.
Newcastle University (UK)
September 2017

Schnorr Non-interactive Zero-Knowledge Proof

Abstract

This document describes the Schnorr non-interactive zero-knowledge (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 that 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 document is not an Internet Standards Track specification; it is published for informational purposes.

This is a contribution to the RFC Series, independently of any other RFC stream. The RFC Editor has chosen to publish this document at its discretion and makes no statement about its value for implementation or deployment. Documents approved for publication by the RFC Editor are not a candidate for any level of Internet Standard; see Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at <http://www.rfc-editor.org/info/rfc8235>.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document.

Table of Contents

1. Introduction	2
1.1. Requirements Language	3
1.2. Notation	3
2. Schnorr NIZK Proof over Finite Field	4
2.1. Group Parameters	4
2.2. Schnorr Identification Scheme	4
2.3. Non-interactive Zero-Knowledge Proof	5
2.4. Computation Cost	6
3. Schnorr NIZK Proof over Elliptic Curve	6
3.1. Group Parameters	6
3.2. Schnorr Identification Scheme	7
3.3. Non-interactive Zero-Knowledge Proof	8
3.4. Computation Cost	8
4. Variants of Schnorr NIZK proof	9
5. Applications of Schnorr NIZK proof	9
6. Security Considerations	10
7. IANA Considerations	11
8. References	11
8.1. Normative References	11
8.2. Informative References	12
Acknowledgements	13
Author's Address	13

1. Introduction

A well-known principle for designing robust public key protocols is 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", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

1.2. Notation

The following notation is 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 $a \mid b$: a divides b
- o $a \parallel b$: concatenation of a and b
- o $[a, b]$: the interval of integers between and including 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., $q \mid p-1$
- o \mathbb{Z}_p^* : a multiplicative group of integers modulo p
- o G_q : a subgroup of \mathbb{Z}_p^* with prime order q
- o g : a generator of G_q
- o g^d : g raised to the power of d
- o $a \bmod b$: a modulo b
- o \mathbb{F}_p : a finite field of p elements, where p is a prime

- o $E(\mathbb{F}_p)$: an elliptic curve defined over \mathbb{F}_p
- o G : a generator of the subgroup over $E(\mathbb{F}_p)$ with prime order n
- o n : the order of G
- o h : the cofactor of the subgroup generated by G , which is equal to the order of the elliptic curve divided by n
- o $P \times [b]$: multiplication of a point P with a scalar b over $E(\mathbb{F}_p)$

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 $q \mid p-1$. Let G_q denote the subgroup of \mathbb{Z}_p^* of prime order q , and g be a generator for the subgroup. Refer to the DSA examples in the NIST Cryptographic Toolkit [NIST_DSA] 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 $A = g^a \bmod p$, where a 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 $V = g^v \bmod p$. 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 $r = v - a * c \bmod q$ 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 A is within $[1, p-1]$ and $A^q = 1 \bmod p$;
2. To verify $V = g^r * A^c \bmod p$.

The first check ensures that A is a valid public key, hence the discrete logarithm of A 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 A is within $[2, p-1]$ instead of $[1, p-1]$.

The process is summarized in the following diagram.

Alice -----	Bob -----
choose random v from $[0, q-1]$	
compute $V = g^v \bmod p$	-- $V \rightarrow$
compute $r = v - a * c \bmod q$	-- c -- choose random c from $[0, 2^t-1]$
	-- $b \rightarrow$ check 1) A is a valid public key 2) $V = g^r * A^c \bmod p$

Information Flows in Schnorr Identification Scheme over Finite Field

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 $c = H(g || V || A || \text{UserID} || \text{OtherInfo})$, 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, or SHA3-512. The bit length of the hash output should be at least equal to that of the order q of the considered subgroup.

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. OtherInfo may contain multiple subitems. In that case, the same rule shall apply to ensure a clear boundary between adjacent subitems.

2.4. Computation Cost

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

To generate a Schnorr NIZK proof, the cost is roughly one modular exponentiation: that is to compute $g^v \bmod p$. In practice, this exponentiation may be precomputed in the offline 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 $A^q \bmod p$ and the other for computing $g^r \cdot A^c \bmod p$. (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 $E(\mathbb{F}_p)$ be an elliptic curve defined over a finite field \mathbb{F}_p , where p is a large prime. Let G be a base point on the curve that serves as a generator for the subgroup over $E(\mathbb{F}_p)$ 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]. Data 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 $A = G \times [a]$, where a 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 $V = G \times [v]$. 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 $r = v - a * c \bmod n$ 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 A is a valid point on the curve and $A \times [h]$ is not the point at infinity;
2. To verify $V = G \times [r] + A \times [c]$.

The first check ensures that A is a valid public key, hence the discrete logarithm of A 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.

Alice	Bob
-----	-----
choose random v from $[1, n-1]$	
compute $V = G \times [v]$	-- $V \rightarrow$
compute $r = v - a * c \bmod n$	-- c -- choose random c from $[0, 2^t-1]$
	-- $b \rightarrow$ check 1) A is a valid public key
	2) $V = G \times [r] + A \times [c]$

Information Flows in Schnorr Identification Scheme over Elliptic Curve

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 $c = H(G || V || A || \text{UserID} || \text{OtherInfo})$, 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 $A = G \times [a]$ with respect to base G over the elliptic curve, Alice generates a Schnorr NIZK proof that contains: $\{\text{UserID}, \text{OtherInfo}, V = G \times [v], r = v - a * c \bmod n\}$, where $c = H(G || V || A || \text{UserID} || \text{OtherInfo})$.

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

To verify the Schnorr NIZK proof in the EC setting, the cost is approximately one multiplication over the elliptic curve: i.e., computing $G \times [r] + A \times [c]$ (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, r) (along with UserID and OtherInfo), and the verifier first computes c , and then checks for $V = g^r * A^c \bmod p$. This requires the transmission of an element V of Z_p , whose size is typically between 2048 and 3072 bits, and an element r of Z_q whose size is typically between 224 and 256 bits. It is possible to reduce the amount of transmitted data to two elements of Z_q as below.

In the modified variant, the prover works exactly the same as before, except that it sends (c, r) instead of (V, r) . The verifier computes $V = g^r * A^c \bmod p$ and then checks whether $H(g || V || A || \text{UserID} || \text{OtherInfo}) = c$. The security of this modified variant follows from the fact that one can compute V from (c, r) and c from (V, r) . Therefore, sending (c, r) is equivalent to sending (V, c, r) , which in turn is equivalent to sending (V, r) . 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, r) with (c, r) , but the benefit is extremely 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 Certification 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, in 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 of 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 $G \times [v]$) 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 $V = g^v \bmod p$ 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, r) and (V, c', r') , based on which he can compute the secret key a by:

$$(r-r')/(c'-c) = (v-a*c-v+a*c')/(c'-c) = a \bmod q.$$

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, r) and (V', c', r') . He can compute the secret key a by:

$$(r-r'+w)/(c'-c) = (v-a*c-v-w+a*c'+w)/(c'-c) = a \bmod q.$$

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 in 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 in OtherInfo whenever necessary.

7. IANA Considerations

This document does not require any IANA actions.

8. References

8.1. Normative References

- [ABM15] Abdalla, M., Benhamouda, F., and P. MacKenzie, "Security of the J-PAKE Password-Authenticated Key Exchange Protocol", 2015 IEEE Symposium on Security and Privacy, DOI 10.1109/sp.2015.41, 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, DOI 10.1007/3-540-44750-4_19, 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, DOI 10.1007/3-540-47721-7_12, 1986.

- [MOV96] Menezes, A., Oorschot, P., and S. Vanstone, "Handbook of Applied Cryptography", 1996.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [SEC1] "Standards for Efficient Cryptography. SEC 1: Elliptic Curve Cryptography", SECG SEC1-v2, May 2009, <<http://www.secg.org/sec1-v2.pdf>>.
- [Stinson06] Stinson, D., "Cryptography: Theory and Practice", 3rd Edition, CRC, 2006.

8.2. Informative References

- [FIPS186-4] National Institute of Standards and Technology, "Digital Signature Standard (DSS)", FIPS PUB 186-4, DOI 10.6028/NIST.FIPS.186-4, July 2013, <<http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf>>.
- [Hao10] Hao, F., "On Robust Key Agreement Based on Public Key Authentication", 14th International Conference on Financial Cryptography and Data Security, DOI 10.1007/978-3-642-14577-3_33, February 2010.
- [HR08] Hao, F. and P. Ryan, "Password Authenticated Key Exchange by Juggling", Lecture Notes in Computer Science, pp. 159-171, from 16th Security Protocols Workshop (SPW'08), DOI 10.1007/978-3-642-22137-8_23, 2011.
- [NIST_DSA] NIST Cryptographic Toolkit, "DSA Examples", <http://csrc.nist.gov/groups/ST/toolkit/documents/Examples/DSA2_All.pdf>.

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 pointed out the optimization technique and 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).

Author's Address

Feng Hao (editor)
Newcastle University (UK)
Urban Sciences Building, School of Computing, Newcastle University
Newcastle Upon Tyne
United Kingdom

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

