Internet Engineering Task Force
Internet-Draft
Intended status: Informational
Expires: June 18, 2014

F. Hao, Ed.
Newcastle University (UK)

December 15, 2013

Schnorr NIZK Proof: Non-interactive Zero Knowledge Proof for Discrete Logarithm
draft-hao-schnorr-01

Abstract

This document describes the 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.

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 June 18, 2014.

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 (i.e., 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.

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
- p: a large prime
- q: a large prime divisor of 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^r: the base g raised to the power of r
- a mod b: a modulo b
- a * b: a multiplied by b
- a || b: concatenation of a and b
- t: the bit length of the challenge chosen by Bob
- H: a secure cryptographic hash function

2. Group parameters

The Schnorr NIZK proof uses exactly the same group setting as DSA (or ECDSA). For simplicity, this document will only describe the Schnorr NIZK proof in the DSA-like group setting. The technique works basically the same in the ECDSA-like group, except that the underlying multiplicative cyclic group is replaced by an additive cyclic group defined over some elliptic curve.

The DSA group setting consists of three parameters: (p, q, g). NIST has published a set of values for (p, q, g) for different security levels; they can be found in Appendix A. The same values can be used for the Schnorr NIZK proof.

3. Schnorr Identification Scheme

The Schnorr identification scheme is a zero-knowledge proof primitive that allows one to prove the knowledge of the discrete logarithm

without leaking any information about its value. It has been proven secure assuming that the verifier is honest and that the discrete logarithm problem is intractable [Stinson06]. This scheme runs interactively between Alice (prover) and Bob (verifier). In the setup of the identification scheme, Alice has published 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 thee 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 h uniformly at random from [0, 2^t - 1], where t is the bit length of the challenge (say t = 80). Bob sends h to Alice.

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

At the end of the protocol, Bob checks if the following equality holds: V = g^b * X^h 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*h mod q  <- h -- choose random h from [0, 2^t-1]

                         -- b -> check if V = g^b * X^h mod p?
```

4. Schnorr NIZK 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 h = H(g || g^v || g^x || UserID || OtherInfo), where UserID is a unique identifier for the prover and OtherInfo is optional data. The OtherInfo is included here for generality, as 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 will 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.

5. Computation cost of Schnorr NIZK proof

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*h mod q}, where h = 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 can 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^h 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^h 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 co-factor being very small (see [MOV96]).

6. 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 registration phrase. Accordingly, the OtherInfo may 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 essentially no different from a self-signed Certificate Signing Request generated by using DSA (or ECDSA). The details are however beyond the scope of this document.

7. 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 only 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 herself, 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 herself (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 her own. Depending the context of specific protocols, other forms of replay attacks should be considered, and appropriate contextual information included into OtherInfo whenever necessary. The UserID and items (if any) in OtherInfo shall be left to specific protocols to define.

8. IANA Considerations

This document has no actions for IANA.

9. Acknowledgements

The editor of this document would like to thank Dylan Clarke, Robert Ransom and Siamak F. Shahandashti for useful comments. This work was supported by the EPSRC First Grant EP/J011541/1.

10. References

10.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[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.

10.2. Informative References

[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.

Appendix A. Group parameters

The Schnorr NIZK proof operates in exactly the same cyclic groups as the Digital Signature Algorithm (DSA). The following 1024-, 2048-, 3072-bit groups are public domain parameters published by NIST for the DSA. They are taken from the following address:

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

1. 1024-bit p and 160-bit q

p =
E0A67598 CD1B763B C98C8ABB 333E5DDA 0CD3AA0E 5E1FB5BA 8A7B4EAB
C10BA338 FAE06DD4 B90FDA70 D7CF0CB0 C638BE33 41BEC0AF 8A7330A3
307DED22 99A0EE60 6DF03517 7A239C34 A912C202 AA5F83B9 C4A7CF02
35B5316B FC6EFB9A 24841125 8B30B839 AF172440 F3256305 6CB67A86
1158DDD9 0E6A894C 72A5BBEF 9E286C6B

q =
E950511E AB424B9A 19A2AEB4 E159B784 4C589C4F

g =
D29D5121 B0423C27 69AB2184 3E5A3240 FF19CACC 792264E3 BB6BE4F7
8EDD1B15 C4DFF7F1 D905431F 0AB16790 E1F773B5 CE01C804 E509066A
9919F519 5F4ABC58 189FD9FF 987389CB 5BEDF21B 4DAB4F8B 76A055FF

E2770988 FE2EC2DE 11AD9221 9F0B3518 69AC24DA 3D7BA870 11A701CE
8EE7BFE4 9486ED45 27B7186C A4610A75

2. 2048-bit p and 224-bit q

p =
C196BA05 AC29E1F9 C3C72D56 DFFC6154 A033F147 7AC88EC3 7F09BE6C
5BB95F51 C296DD20 D1A28A06 7CCC4D43 16A4BD1D CA55ED10 66D438C3
5AEBAABF 57E7DAE4 28782A95 ECA1C143 DB701FD4 8533A3C1 8F0FE235
57EA7AE6 19ECACC7 E0B51652 A8776D02 A425567D ED36EABD 90CA33A1
E8D988F0 BBB92D02 D1D20290 113BB562 CE1FC856 EEB7CDD9 2D33EEA6
F410859B 179E7E78 9A8F75F6 45FAE2E1 36D252BF FAFF8952 8945C1AB
E705A38D BC2D364A ADE99BE0 D0AAD82E 53201214 96DC65B3 930E3804
7294FF87 7831A16D 5228418D E8AB275D 7D75651C EFED65F7 8AFC3EA7
FE4D79B3 5F62A040 2A111759 9ADAC7B2 69A59F35 3CF450E6 982D3B17
02D9CA83

q =
90EAF4D1 AF0708B1 B612FF35 E0A2997E B9E9D263 C9CE6595 28945C0D

g =
A59A749A 11242C58 C894E9E5 A91804E8 FA0AC64B 56288F8D 47D51B1E
DC4D6544 4FECA011 1D78F35F C9FDD4CB 1F1B79A3 BA9CBEE8 3A3F8110
12503C81 17F98E50 48B089E3 87AF6949 BF8784EB D9EF4587 6F2E6A5A
495BE64B 6E770409 494B7FEE 1DBB1E4B 2BC2A53D 4F893D41 8B715959
2E4FFFDF 6969E91D 770DAEBD 0B5CB14C 00AD68EC 7DC1E574 5EA55C70
6C4A1C5C 88964E34 D09DEB75 3AD418C1 AD0F4FDF D049A955 E5D78491
C0B7A2F1 575A008C CD727AB3 76DB6E69 5515B05B D412F5B8 C2F4C77E
E10DA48A BD53F5DD 498927EE 7B692BBB CDA2FB23 A516C5B4 533D7398
0B2A3B60 E384ED20 0AE21B40 D273651A D6060C13 D97FD69A A13C5611
A51B9085

3. 2048-bit p, 256-bit q

p =
F56C2A7D 366E3EBD EAA1891F D2A0D099 436438A6 73FED4D7 5F594959
CFFEBCA7 BE0FC72E 4FE67D91 D801CBA0 693AC4ED 9E411B41 D19E2FD1
699C4390 AD27D94C 69C0B143 F1DC8893 2CFE2310 C8864120 47BD9B1C
7A67F8A2 59091326 27F51A0C 866877E6 72E55534 2BDF9355 347DBD43
B47156B2 C20BAD9D 2B071BC2 FDCF9757 F75C168C 5D9FC431 31BE162A
0756D1BD EC2CA0EB 0E3B018A 8B38D3EF 2487782A EB9FBF99 D8B30499
C55E4F61 E5C7DCEE 2A2BB55B D7F75FCD F00E48F2 E8356BDB 59D86114
028F67B8 E07B1277 44778AFF 1CF1399A 4D679D92 FDE7D941 C5C85C5D
7BFF91BA 69F9489D 531D1EBF A727CFDA 651390F8 021719FA 9F7216CE
B177BD75

q =

C24ED361 870B61E0 D367F008 F99F8A1F 75525889 C89DB1B6 73C45AF5
867CB467

g =
8DC6CC81 4CAE4A1C 05A3E186 A6FE27EA BA8CDB13 3FDCE14A 963A92E8
09790CBA 096EAA26 140550C1 29FA2B98 C16E8423 6AA33BF9 19CD6F58
7E048C52 666576DB 6E925C6C BE9B9EC5 C16020F9 A44C9F1C 8F7A8E61
1C1F6EC2 513EA6AA 0B8D0F72 FED73CA3 7DF240DB 57BBB274 31D61869
7B9E771B 0B301D5D F0595542 5061A30D C6D33BB6 D2A32BD0 A75A0A71
D2184F50 6372ABF8 4A56AEEE A8EB693B F29A6403 45FA1298 A16E8542
1B2208D0 0068A5A4 2915F82C F0B858C8 FA39D43D 704B6927 E0B2F916
304E86FB 6A1B487F 07D8139E 428BB096 C6D67A76 EC0B8D4E F274B8A2
CF556D27 9AD267CC EF5AF477 AFED029F 485B5597 739F5D02 40F67C2D
948A6279

4. 3072-bit p, 256-bit q

p =
90066455 B5CFC38F 9CAA4A48 B4281F29 2C260FEE F01FD610 37E56258
A7795A1C 7AD46076 982CE6BB 956936C6 AB4DCFE0 5E678458 6940CA54
4B9B2140 E1EB523F 009D20A7 E7880E4E 5BFA690F 1B9004A2 7811CD99
04AF7042 0EEFD6EA 11EF7DA1 29F58835 FF56B89F AA637BC9 AC2EFAAB
90340222 9F491D8D 3485261C D068699B 6BA58A1D DBBEF6DB 51E8FE34
E8A78E54 2D7BA351 C21EA8D8 F1D29F5D 5D159394 87E27F44 16B0CA63
2C59EFD1 B1EB6651 1A5A0FBF 615B766C 5862D0BD 8A3FE7A0 E0DA0FB2
FE1FCB19 E8F9996A 8EA0FCCD E5381752 38FC8B0E E6F29AF7 F642773E
BE8CD540 2415A014 51A84047 6B2FCEB0 E388D30D 4B376C37 FE401C2A
2C2F941D AD179C54 0C1C8CE0 30D460C4 D983BE9A B0B20F69 144C1AE1
3F9383EA 1C08504F B0BF3215 03EFE434 88310DD8 DC77EC5B 8349B8BF
E97C2C56 0EA878DE 87C11E3D 597F1FEA 742D73EE C7F37BE4 3949EF1A
0D15C3F3 E3FC0A83 35617055 AC91328E C22B50FC 15B941D3 D1624CD8
8BC25F3E 941FDDC6 20068958 1BFEC416 B4B2CB73

q =
CFA0478A 54717B08 CE64805B 76E5B142 49A77A48 38469DF7 F7DC987E
FCCFB11D

g =
5E5CBA99 2E0A680D 885EB903 AEA78E4A 45A46910 3D448EDE 3B7ACCC5
4D521E37 F84A4BDD 5B06B097 0CC2D2BB B715F7B8 2846F9A0 C393914C
792E6A92 3E2117AB 805276A9 75AADB52 61D91673 EA9AAFFE ECBFA618
3DFCB5D3 B7332AA1 9275AFA1 F8EC0B60 FB6F66CC 23AE4870 791D5982
AAD1AA94 85FD8F4A 60126FEB 2CF05DB8 A7F0F09B 3397F393 7F2E90B9
E5B9C9B6 EFEF642B C48351C4 6FB171B9 BFA9EF17 A961CE96 C7E7A7CC
3D3D03DF AD1078BA 21DA4251 98F07D24 81622BCE 45969D9C 4D6063D7
2AB7A0F0 8B2F49A7 CC6AF335 E08C4720 E31476B6 7299E231 F8BD90B3
9AC3AE3B E0C6B6CA CEF8289A 2E2873D5 8E51E029 CAFBD55E 6841489A
B66B5B4B 9BA6E2F7 84660896 AFF387D9 2844CCB8 B6947549 6DE19DA2

E58259B0 90489AC8 E62363CD F82CFD8E F2A427AB CD65750B 506F56DD
E3B98856 7A88126B 914D7828 E2B63A6D 7ED0747E C59E0E0A 23CE7D8A
74C1D2C2 A7AFB6A2 9799620F 00E11C33 787F7DED 3B30E1A2 2D09F1FB
DA1ABBBF BF25CAE0 5A13F812 E34563F9 9410E73B

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)192-208-6384
EMail: feng.hao@ncl.ac.uk