Secure Comparator:
a ZKP-Based Authentication System

Revision 1.2

Ignat Korchagin
ignat@cossacklabs.com

Eugene Pilyankevich
eugene@cossacklabs.com

ABSTRACT

This paper presents Secure Comparator, a way to implement Zero Knowledge Proof algorithm called Socialist Millionaire's Problem, to compare secrets between two parties. Compared to existing implementations, Secure Comparator provides better security guarantees, stronger cryptographic math, and, possibly, more integration-friendly architecture.

Keywords

Zero Knowledge Proof, Socialist Millionaire's Problem, authentication, passwords, ed25519, Elliptic Curve Cryptography.

Update note: Revision 1.2 addresses dishonest prover problem, pointed out by community feedback, in section 5.3.

1. Introduction.

The password is the oldest and the most widely used pillar of authentication. In the modern technologically heterogeneous and distributed environment, password-based authentication is frequently the only available method to prove identity to a third party.

Being a secret, it's best protected when it never leaves the safe zone. Proving identity involves communicating the secret to another party, which eventually exposes the whole or a part of the secret, in a direct or indirect (hashed) form. In the real world, authentication traffic passes thousands of systems between a prover (you) and a verifier (an entity which eventually decides whether your secret is valid and you deserve the privileges you claim).

While communicating it, you open the secret to various threat vectors, which expose it to attackers in one or another way.

2. Existing Authentication Methods and Motivation

Existing authentication methods provide some levels of protection, but each of them has significant drawbacks.

So far, most security systems have used only three types of cryptographic primitives: encryption, key agreement, and digital signatures. Authentication secret security is achieved by combining some of this primitives in a protocol.

Over time, each authentication method in Internet systems was exploited in some way, and required new techniques to provide better security guarantees and mitigate attacks:
- for plaintext passwords all kinds of passive traffic interception attacks were used;
- once parties started exchanging hashes, dictionary attacks and active browser attacks (using a hash to fake authentication handshake) became used;
- once parties started to use strong authentication, which was based on key agreement algorithms, man-in-the-middle attacks to degrade ciphers became used;

As history evolves, more and more modern data protection techniques fail against sophisticated attacks. Wouldn't it be great to avoid transmitting passwords at all?

3. Zero Knowledge Proof

Basing on problems above, we propose building secret-based authentication around Zero Knowledge Proof: a method for one party (the prover) to prove to another party (the verifier) that some statement is true with following properties [1]:
1. *Completeness*: if the statement is true, the honest verifier (that is, one following the protocol properly) will be convinced of this fact by a honest prover.
2. *Soundness*: if the statement is false, no cheating prover can convince the honest verifier that it is true, except with some small probability.
3. *Zero-knowledge*: if the statement is true, no cheating verifier learns anything other than this fact. This is formalized by showing that every cheating verifier has some simulator that, given only the statement to be proved (and no access to the prover), can produce a transcript that "looks like" an interaction between the honest prover and the cheating verifier.

The first two properties form the basis of authentication and can be achieved by a proper combination of other cryptographic primitives. The last property ensures the protocol flow and outcome is meaningful only to honest protocol participants and nobody else. In other words, the protocol by design does not leak any auxiliary information to the third-party observers or even the verifier.

3.1 Socialist Millionaire's Problem (SMP Protocol)

Socialist Millionaire's Problem allows two parties to verify whether the secret they use is identical without allowing either party to learn anything else about the other's secret value. That is, if communicating parties' secrets do not match, no party learns anything more that this fact.

Zero Knowledge Proofs have been scientifically studied and verified for two decades now. We've picked Socialist Millionaire's Problem for its simplicity, and good track record in OTR protocol, where it is used to authenticate remote parties.

3.2 Benefits of SMP

In short, the SMP usage allows achieving the following benefits:

- a new mechanism for HTTP password authentication (ensures that password or hashes of the password never gets sent over the wire);
- a new way to confirm established secret key after the key agreement;
- a mutual authentication mechanism;
- remote attestation mechanism;
- an insider-resistant OTP reinforcement.

4. Secure Comparator

Secure Comparator is an SMP-based authentication method, which we use to compare secrets like passwords and access tokens. We have strived to get better security guarantees, than existing SMP implementations, and more flexibility.

4.1 Hardening SMP

To build authentication scheme around SMP, we have used OTR protocol as a reference.

However, OTR uses 1536-bit group algebra as the basis of their computations. Since SMP is very similar in operations to Diffie-Hellman key exchange, it is subject to almost same security considerations. The recent disclosure of the LogJam attack on TLS protocol [4], which targets Diffie-Hellman key exchange, provides proof that 512-bit groups might be practically cracked today using number field sieve algorithm.

The same paper provides estimates that 1024-bit groups may already be vulnerable now or will become vulnerable sometime in the near future. Based on the above we feel that 1536-bit might not provide adequate security level. In addition, we preferred to have a more fundamental solution to this than just increasing the field bit size, which puts extra pressure on both algorithm speed and memory requirements. Therefore, we decided to harden the SMP by re-implementing it based on Elliptic Curve Cryptography.

Both SMP and DH security is mostly based on discrete logarithm problem (DLP): finding an exponent used to compute the power of some base in a cyclic group does not have algorithms with polynomial time complexity. However, in ECC domain this problem is considered even more secure, because, unlike in prime-field algebra, where such algorithms exist with sub-exponential time complexity, only exponential time algorithms exist for ECC.

To make SMP (as well as most of the DLP-based algorithms) ECC-enabled, following has to be changed in the algorithm flow:

- choose target ECC domain parameters;

- introduce deterministic mapping of secret and random protocol values to big integers greater than 0 and smaller than ECC field base point order;
- replace all prime-field group multiplications with ECC point additions;
- replace all prime-field group divisions with ECC point subtractions;
- replace all prime-filed group exponentiations with ECC point on big integer multiplications.

After considering all possible variants for a good ECC domain, we decided to use ed25519, because:

- we wanted something better protected from side-attacks than conventional NIST curves;
- fast and performant;
- available in public domain, and, preferably, coming from trusted experts (Daniel J. Bernstein);
- has little or no weak private keys, so above mapping can be designed easily (a simple hash function with truncated output).

However, original ed25519 implementation lacked some basic ECC primitives, so it had to be extended for SMP.

4.2 Extending ed25519

Even though ed25519 is a good candidate, it is a digital signature algorithm. In SMP, we need a different combination of ECC primitive operations and as a consequence - different algorithm parameter handling.

In addition, basic ed25519 boasts constant time operations protecting users' secrets from side-channel (mostly timing) attacks. Since the code was written with high optimizations in mind, ed25519 had a limited subset of operations to start with:

- $Q = d * G$ - basic ECC base point multiplication;
- $R = s * G + r * Q$ - sum of two points, where one is a multiple of base point.

SMP, on the other hand, requires $T = u * P$ - simple point multiplication, but where P is some random (not known in advance) point. First two obvious choices were not the right ones to pick:

1) To implement this operation based on existing features of ed25519, the obvious way was to reuse the second function (sum of two points) passing n - ECC base point order - as s parameter. Effectively, since $n * G = O$ (ECC zero point), so $n * G + r * Q = O + r * Q = r * Q$. However, first ed25519 operation is constant time, while the second one is not. For digital signature usage, it does not create serious threat - the second operation is used only to verify signatures with the public key (no secret information is used). In our case, our adapted function may be used to multiply some secret random value on an arbitrary point.

2) The reusing approach, which was taken in the first ed25519 operation (scalar multiplication replaced by point addition from a table of a precomputed point factors) also made little sense, since its constant time properties are based on a large precomputed table (~30 KB) specifically for G (which is constant and known in advance). We cannot afford to do the same computations in the process for arbitrary point, because of performance considerations: since we may receive a random point from our peer we would have to recompute the table each time.

Instead of trying to make the operation constant time, we decided to take the blinding approach. Let's suppose we want to compute $R = d * Q$:

- generate random integer *rnd*
- compute $R = rnd * G + d * Q$ - variable time, where the attacker can only get some information about $rnd + d$, but since they know neither *d* nor *rnd*, it is similar to as if *d* was encrypted by *rnd*
- compute $P = rnd * G$ - constant time operation, available in original ed25519, so attacker gets nothing
- compute $R = R - P = rnd * G + d * Q - rnd * G = d * Q$ - constant time operation (simple point subtraction, does not involve secret handling)

5. The Protocol

Our SMP protocol is very similar to SMP implementation in Cypherpunk's OTR [3] except that we use ECC for all computations.

Let's suppose we have two parties with secrets x and y respectively, and they wish to know whether x == y. They use ed25519 curve with *G* as a basepoint. Alice starts the protocol:

- Alice
  - picks two random numbers: a2 and a3
  - computes G2a = a2 * G and G3a = a3 * G
  - sends G2a and G3a to Bob
- Bob
  - picks two random numbers: b2 and b3
  - computes G2b = b2 * G and G3b = b3 * G
  - computes G2 = b2 * G2a and G3 = b3 * G3a
  - picks random number r
  - computes Pb = r * G3 and Qb = r * G + y * G2
  - sends G2b, G3b, Pb and Qb to Alice
- Alice
  - computes G2 = a2 * G2b and G3 = a3 * G3b
  - picks random number s
  - computes Pa = s * G3 and Qa = s * G + x * G2
  - computes Ra = a3 * (Qa - Qb)
  - sends Pa, Qa, Ra to Bob
- Bob
  - computes Rb = b3 * (Qa - Qb)
  - computes Rab = b3 * Ra
  - checks whether Rab == Pa - Pb
  - sends Rb to Alice
- Alice
  - computes Rab = a3 * Rb
  - checks whether Rab == Pa - Pb

If the *Rab == Pa - Pb* check succeeds then each party is convinced that x == y. Since *Rab = (Pa - Pb) + (a3 * b3 * (x - y)) * G2*, iff *x == y, Rab = (Pa - Pb) + 0 * G2 = Pa - Pb*. If *x≠ y*, then *a3 * b3 * (x - y) * G2* is a random ECC point not known to any party, so no information is revealed.

5.1 ECC-Specific Considerations

All numbers used in ECC calculations for best security must be less than ECC base point order. For ed25519, a suitable number is any 32-byte array, but with three last bits cleared in the first byte, first bit cleared and second bit set in the last byte. This applies to any random numbers used as well as for the secrets themselves.

5.2 Hashing Secrets

We do not directly use secret information in SMP calculations. To allow arbitrary length string to be compared, we hash all the information and compare hashes instead. Currently, we use SHA-256 for that.

5.3 ECC-based non-interactive zero-knowledge proofs of knowledge

To enforce communicating peers strictly follow the protocol and to prevent one of the (possible malicious) parties to falsify the protocol outcome [10] and [3] use non-interactive zero-knowledge proofs of knowledge to create and verify proofs that each intermediate parameter was generated according to protocol.

[10] describes such zero-knowledge proofs of knowledge based on extended version of Schnorr's protocol. Since our ECC-based implementation utilizes ed25519 curve, which is the basis for elliptic curve variant of Shnorr's signature scheme, it is easy to transition those proofs to ECC domain taking into account provisions set in [11], namely:

- all random numbers should be 32 bytes long with three last bits cleared in the first byte, first bit cleared and second bit set in the last byte
- instead of shorter hash functions we use SHA-512
- whole output of hash function is used: hash function output is treated like a big number and reduced by *q = 2^252 + 27742317777372353535851937790883648493* before doing further computations
- signatures should be 64 bytes long with first three bits cleared in the last byte

We describe ECC-variations of used proofs (which mirror their counterparts in [10]) below.

5.3.1 Proof of knowledge of EC discrete logarithm.

Given a curve with generator G and its order *n* Alice can prove to Bob she knows *0 < x < n* such that *Q = x * G* by:

- selecting random integer *r: 0 < r < n*
- computing *W = r * G*, *c = SHA512(W)*, *d = r - xc mod q*
- presenting *(c, d)* to Bob

Bob verifies the proof by checking $c == SHA512(d * G + c * Q)$

5.3.2 Proof of knowledge of EC discrete coordinates.

Given a curve with generators *G1* and *G2* and their order *n* Alice can prove to Bob she knows $0 < x1 < n$ and $0 < x2 < n$ such that $Q = x1 * G1 + x2 * G2$ by:

- selecting random integers $r1, r2: 0 < r1 < n, 0 < r2 < n$
- computing $W = r1 * G1 + r2 * G2, c = SHA512(W), d1 = r1 - x1*c \bmod q, d2 = r2 - x2*c \bmod q$
- presenting *(c, d1, d2)* to Bob

Bob verifies the proof by checking $c == SHA512(d1 * G1 + d2 * G2 + c * Q)$

5.3.3 Proof of equality of two EC discrete logarithms.

Given a curve with generators *G1* and *G2* and their order *n* Alice can prove to Bob she knows $0 < x < n$ such that $Q1 = x * G1$ and $Q2 = x * G2$ by:

- selecting random integer $r: 0 < r < n$
- computing $W1 = r * G1, W2 = r * G2, c = SHA512(W1, W2), d = r - xc \bmod q$
- presenting *(c, d)* to Bob

Bob verifies the proof by checking $c == SHA512(d * G1 + c * Q1, d * G2 + c * Q2)$

[10] describes more non-interactive proofs for different scenarios. Those proofs are not described here, since our ECC SMP protocol mirrors the one described in [3] and is a variant of "version without fairness" ([10]), which does not use the former.

6. Existing ZKP/ZKPP Systems.

Although we believe that our implementation is novel, the idea of using Zero Knowledge Proofs for password authentication (which is called ZKPP) is not. Below are the two best systems we're aware of, and our explanation what we differ with.

Lightweight Zero Knowledge Proof Authentication [5]

The paper describes a classic zero knowledge protocol (based on graph isomorphism) in the application within web authentication. Unlike SMP, the protocol is a one-way client-server authentication mechanism. The downside of this approach is having soundness error probability of 1/2 for one round of challenge-response. Therefore, in order to lower it many iterations of the protocol are required. This significantly increases the number of round-trips in overall authentication scheme (the paper mentions ~2000).

RFC 2945: SRP Authentication and Key Exchange System [6]

Secure remote password protocol was developed primarily to support zero knowledge password verification. It also uses DLP and public key cryptography for its operations.

Unlike SMP, its use-case (client-server password verification) is embedded in the protocol flow. The advantage of this is that server does not have to store plaintext password to do the protocol. However, it provides a one-way authentication mechanism, whereas SMP implicitly does mutual authentication. From cryptography perspective, SRP requires explicit salt to protect from replay attack, where SMP has such protection inherently from its flow. Also, because of extended required cryptographic primitives (need to do field element multiplication), it is hard to port SRP to ECC domain.

7. Other uses

Password authentication is only one narrow use-case; however, being the most obvious and demanding, it is presented as the main purpose here. Using Secure Comparator-based authentication is not limited to it though, as the system will work with any comparison of secrets, which have public identifiers (usernames, customer names, record IDs) and secret parts.

One of the obvious use-cases is enabling two personal data-processing systems to compare secret customer identifiers (SSN, Tax ID) between parties without established trust relationships, which can be essential in interactions, where request query has parts, the disclosure of which can affect privacy or security.

8. Example Implementation

We have implemented Secure Comparator in our open-source security services library, Themis [7]. Feel free to explore the source code on our GitHub [8], or read the blog post outlining various practical aspects of using Secure Comparator [9].

References

[1] ftp://ftp.inf.ethz.ch/pub/crypto/publications/Maurer09.pdf
[2] http://markus-jakobsson.com/papers/jakobsson-crypto96.pdf
[3] Off-the-Record Messaging Protocol version 3, https://otr.cypherpunks.ca/Protocol-v3-4.0.0.html
[4] https://weakdh.org/imperfect-forward-secrecy-ccs15.pdf
[5] http://cs.nyu.edu/~zaremba/docs/zkp.pdf
[6] https://en.wikipedia.org/wiki/Secure_Remote_Password_protocol
[7] https://www.cossacklabs.com/themis
[8] https://github.com/cossacklabs/themis/tree/master/src/themis
[9] https://cossacklabs.com/introducing_secure_comparator.html
[10] https://www.win.tue.nl/~berry/papers/dam.pdf
[11] http://ed25519.cr.yp.to/ed25519-20110926.pdf