Crypto Forum Research Group
Internet-Draft
Intended status: Informational
Expires: April 4, 2019

H. Krawczyk
IBM Research
October 01, 2018

The OPAQUE Asymmetric PAKE Protocol
draft-krawczyk-cfrg-opaque-00

Abstract

This draft describes the OPAQUE protocol, a secure asymmetric password authenticated key exchange (aPAKE) that supports mutual authentication in a client-server setting without any reliance on PKI. OPAQUE is the first PKI-free aPAKE to accommodate secret salt and therefore is the first to be secure against pre-computation attacks upon server compromise. In contrast, prior aPAKE protocols did not use salt and if they did, the salt was transmitted in the clear from server to user allowing for the building of targeted pre-computed dictionaries. OPAQUE security has been proven by Jarecki et al. (Eurocrypt 2018) in a strong and universally composable formal model of aPAKE security. In addition, the protocol provides forward secrecy and the ability to hide the password from the server even during password registration.

Strong security, good performance and an array of additional features make OPAQUE a natural candidate for practical use and for adoption as a standard. To this end, this draft presents several optimized instantiations of OPAQUE and ways of integrating OPAQUE with TLS.

1. Introduction

Password authentication is the prevalent form of authentication in the web and in most other applications. In the most common implementation, a user authenticates to a server by entering its user id and password where both values are transmitted to the server under the protection of TLS. This makes the password vulnerable to TLS failures, including many forms of PKI attacks, certificate mishandling, termination outside the security perimeter, visibility to middle boxes, and more. Moreover, even under normal operation, passwords are always visible in plaintext form at the server upon TLS decryption.

Asymmetric (or augmented) Password Authenticated Key Exchange (aPAKE) protocols are designed to provide password authentication and mutually authenticated key exchange without relying on PKI (except during user/password registration) and without disclosing passwords to servers or other entities other than the client machine. A secure aPAKE should provide the best possible security for a password protocol, namely, it should only be open to inevitable attacks: Online impersonation attempts with guessed user passwords and offline dictionary attacks upon the compromise of a server and leakage of its "password file". In the latter case, the attacker learns a mapping of a user's password under a one-way function and uses such a mapping to validate potential guesses for the password. Crucially important is for the password protocol to use an unpredictable one-way mapping or otherwise the attacker can pre-compute a deterministic list of mapped passwords leading to almost instantaneous leakage of passwords upon server compromise.

Quite surprisingly, in spite of the existence of multiple designs for (PKI-free) aPAKE protocols, none of these protocols is secure against pre-computation attacks. In particular, none of these protocols can use the standard technique against pre-computation that combines _secret_ random values ("salt") into the one-way password mappings. Either these protocols do not use salt at all or, if they do, they transmit the salt from server to user in the clear, hence losing the secrecy of the salt and its defense against pre-computation. Furthermore, the transmission of salt often incurs in additional protocol messages.

This draft describes OPAQUE, the first PKI-free secure aPAKE that is secure against pre-computation attacks and capable of using secret salt. OPAQUE has been recently defined and studied by Jarecki et al. [OPAQUE] who prove the security of the protocol in a strong aPAKE model that ensures security against pre-computation attacks and is formulated in the Universal Composability framework [Canetti01] under the random oracle model. In contrast, very few aPAKE protocols have been proven formally and those proven were analyzed in a weak security model that allows for pre-computation attacks (e.g., [GMR06]). This is not just a formal issue: these protocols are actually vulnerable to such attacks! Furthermore, as far as we know, none of the protocols discussed recently as candidates for standardization (e.g., SPAKE2+ [I-D.irtf-cfrg-spake2] and AugPAKE [RFC6628]) enjoys a proof of security, not even in a weak model. The same holds for the SRP protocol [RFC2945] and none of these protocols accommodates secret salt.

OPAQUE's design is based on the seminal work of Ford and Kaliski [FK00] with variants studied by Boyen [Boyen09] and Jarecki et al. [JKKX16], although none of these papers presented a proof of aPAKE security (not even in a weak model).

In addition to its proven security against pre-computation attacks, OPAQUE's security features include forward secrecy (essential for protecting past communications in case of password leakage) and the ability to hide the password from the server even during password registration. Moreover, good performance and an array of additional features make OPAQUE a natural candidate for practical use and for adoption as a standard. Such features include the ability to increase the difficulty of offline dictionary attacks via iterated hashing and offloading these iterations to the client, extensibility of the protocol to support storage and retrieval of user's secrets solely based on a password, and being amenable to a multi-server distributed implementation where offline dictionary attacks are not possible without breaking into a threshold of servers (such distributed solution requires no change or awareness on the client side relative to a single-server implementation).

OPAQUE is defined and proven as the composition of two functionalities: An Oblivious PRF (OPRF) and a key-exchange protocol.

It can be seen as a "compiler" for transforming any key-exchange protocol (with KCI security - see below) into a secure aPAKE protocol. In OPAQUE, the user stores a secret private key at the server during password registration and retrieves this key each time it needs to authenticate to the server. The OPRF security properties ensure that only the correct password can unlock the private key while at the same time avoiding potential offline guessing attacks. This general composability property provides great flexibility and enables a variety of OPAQUE instantiations, from optimized performance to integration with TLS. The latter aspect is of prime importance as the use of OPAQUE with TLS constitutes a major security improvement relative to the standard password-over-TLS practice. At the same time, the combination with TLS builds OPAQUE as a fully functional secure communications protocol and can help provide privacy to account information sent by the user to the server prior to authentication.

The KCI property required from KE protocols for use with OPAQUE states that knowledge of a party's private key does not allow an attacker to impersonate others to that party. This is an important security property achieved by most public-key based KE protocols, including protocols that use signatures or public key encryption for authentication. It is also a property of many implicitly authenticated protocols (e.g., HMQV) but not all of them. We also note that key exchange protocols based on shared keys do not satisfy the KCI requirement, hence they are not considered in the OPAQUE setting.

This draft defines OPAQUE with a specific, efficient instantiation over elliptic curves of the OPRF component and with a few KE schemes, including the HMQV [HMQV] and SIGMA [SIGMA] protocols, as well as several suggestions for integrating OPAQUE with TLS 1.3 [I-D.ietf-tls-tls13] offering different tradeoffs between simplicity, performance and user privacy.

The computational cost of OPAQUE is determined by the cost of the OPRF, the cost of a regular Diffie-Hellman exchange, and the cost of authenticating such exchange. In our elliptic-curve implementation of the OPRF, the cost for the client is two exponentiations (one or two of which can be fixed base) and one hashing-into-curve operation [I-D.irtf-cfrg-hash-to-curve]; for the server, it is just one exponentiation. The cost of a Diffie-Hellman exchange is as usual two exponentiations per party (one of which is fixed-base). Finally, the cost of authentication per party depends on the specific KE protocol: it is just 1/6 of an exponentiation with HMQV and it is one signature in the case of SIGMA and TLS 1.3. These instantiations preserve the number of messages (two or three) in the underlying KE protocol except in one of the TLS instantiations where user privacy requires an additional round trip.

1.1. Terminology

In this document, the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" are to be interpreted as described in BCP 14, RFC 2119 [RFC2119]

1.2. Notation

Throughout this document the first argument to a keyed function represents the key; separated by a semicolon are the function inputs typically implemented as a unambiguous concatenation of strings.

Except if said otherwise, random choices in this specification refer to drawing with uniform distribution from a given set (i.e., "random" is short for "uniformly random").

The name OPAQUE: A homonym of O-PAKE where O is for Oblivious (the name OPAKE was taken).

2. DH-OPRF

A fundamental piece in the definition of OPAQUE is an Oblivious Pseudo Random Function (OPRF).

An Oblivious PRF (OPRF) is an interactive protocol between a server S and a user U defined by a special pseudorandom function (PRF), denoted F. The server's input to the protocol is a key k for PRF F and the user's input is a value x in the domain of F. At the end of the protocol, U learns F(k;x) and nothing else while S learns nothing from the protocol execution (in particular nothing about x or the value F(k;x)).

OPAQUE uses a specific OPRF instantiation, called DH-OPRF, where the PRF, denoted F, is defined as follows.

Parameters: Hash function H (e.g., a SHA2 or SHA3 function), a cyclic group G of prime order q (with a defined unique string representation of its elements), a generator g of G, and hash function H' mapping arbitrary strings into G (where H' is modeled as a random oracle).

o DH-OPRF domain: Any string

o DH-OPRF range: The range of the hash function H o DH-OPRF key: A random element k in [0..q-1]; denote v=g^k

o DH-OPRF Operation: F(k; x) = H(x, v, H'(x)^k)

Protocol for computing DH-OPRF, U with input x and S with input k:

o U: choose random r in [0..q-1], send a=H'(x)*g^r to S

o S: upon receiving a value a, respond with v=g^k and b=a^k

o U: upon receiving values b and v, set the PRF output to H(x, v, b*v^{-r})

All received values (a, b, v) are checked to be non-unit elements in G. A party aborts if the check fails. In the case of Elliptic Curves this test is typically inexpensive - see [I-D.irtf-cfrg-spake2] for ways to deal with this check (including co-factor exponentiation) that apply to DH-OPRF as well.

Note (exponential blinding): An alternative way of computing DH-OPRF is for U to send a=(H'(x))^r in the first message and set the function output to H(x,v,b^{1/r}) upon receiving S's response. However, note that the multiplicative blinding above is more efficient as the g^r exponentiation uses a fixed base. Moreover, in cases where the user caches v (e.g., for sites it visits often) then one can also optimize the exponentiation v^{-r}.

Note: For elliptic curve implementations of DH-OPRF, the hashing into the curve operation has been studied extensively with known efficient implementations, see [I-D.irtf-cfrg-hash-to-curve].

2.1. Hardening OPRF via user iterations

Protocol OPAQUE can be further strengthened against offline dictionary attacks by applying to the output of DH-OPRF an iterated hash for some number n of iterations. This increases the cost of an offline attack upon the compromise of the server as the attacker will need to perform n iterations for each guess of PwdU it tries to validate. For this purpose we re-define DH-OPRF as F(k;x) = I^n(H(x, v, H'(x)^k)) where I is a specialized hash function designed for hashing passwords such as Argon2 [I-D.irtf-cfrg-argon2] or scrypt [RFC7914]. The symbol I^n denotes n iterations of function I. We note that in OPAQUE, it is the user who performs these iterations. The value n can be a public constant or it can be communicated by the server as part of its OPAQUE message.

3. OPAQUE Specification

OPAQUE consists of the concurrent run of an OPRF protocol and a key-exchange protocol KE (one that provides mutual authentication based on public keys and satisfies the KCI requirement). We first define OPAQUE in a generic way based on any OPRF and any PK-based KE, and later show specific instantiation using DH-OPRF (defined in Section 2) and several KE protocols. The user takes the role of initiator in these protocols and the server the responder's. The private-public keys for the user are denoted PrivU and PubU, and for the server PrivS and PubS.

3.1. Password registration

Password registration is run between a user U and a server S. It is assumed that the user can authenticate the server during this registration phase (this is the only part in OPAQUE that requires some form of authenticated channel, either physical, out-of-band, cryptographic, etc.)

- o U chooses password PwdU and a pair of private-public keys PrivU and PubU for the given protocol KE.

- o S chooses OPRF key kU (random and independent for each user U) and sets vU = g^kU; it also chooses its own pair of private-public keys PrivS and PubS for use with protocol KE (the server can use the same pair of keys with multiple users), and sends PubS to U.

- o U and S run OPRF(kU;PwdU) with only U learning the result, denoted RwdU (mnemonics for "randomized password").

- o U generates an "envelope" EnvU defined as

  EnvU = AuthEnc(RwdU; PrivU, PubU, PubS, vU)

  where AuthEnc is an authenticated encryption function with the "key committing" property (see note below). In EnvU only PrivU requires encryption while all values (except vU) require authentication. PubU can be omitted if it can be reconstructed from PrivU (although it will be generally more efficient to include it under EnvU). vU can be completely omitted from EnvU but then the server will have to send it with its OPRF response in addition to EnvU.

- o U sends EnvU and PubU to S and erases PwdU, RwdU and all keys. S stores (EnvU, PubS, PrivS, PubU, kU, vU) in a user-specific record. If PrivS and PubS are used for different users, they can be stored separately and omitted from the record.

Note (password rules). The above procedure has the significant advantage that the user's password is not disclosed to the server even during registration. Some sites require learning the user's password for enforcing password rules. Doing so voids this important security property of OPAQUE and is not recommended. Moving the password check procedure to the client side is a more secure alternative.

Note (key committing authenticated encryption). The function AuthEnc used to compute EnvU needs to satisfy a property called "key committing". That is, given a pair of random AuthEnc keys, it should be infeasible to create an authenticated ciphertext that successfully decrypts under the two keys. One method is to use encrypt-then-MAC where the MAC is collision resistant with respect to keys, i.e., given two random keys it is hard to find a message that has the same authentication tag under the two keys. In particular, HMAC with an output of 256 or more bits has this property.

Note (salt). We note that in OPAQUE the OPRF key acts as the secret salt value that ensures the infeasibility of pre-computation attacks.

3.2. Online OPAQUE protocol

After registration, the user and server can run the OPAQUE protocol as a password-authenticated key exchange. The protocol consists of:

- transmitting user/account information to the server so that the server can retrieve the user's record;
- OPRF execution between user and server through which the user obtains the value RwdU;
- the sending of EnvU from server to user;
- decryption by the user of EnvU using RwdU to obtain the user's private and public key as well as the authenticated server's public key;
- use of the public and private keys of each party to run the specified KE protocol.

OPAQUE is optimized by running the OPRF and KE concurrently with interleaved and combined messages (while preserving the internal ordering of messages in each protocol). In all cases, the user needs to obtain RwdU and EnvU before it can use its own private key PrivU and the server's public key PubS in the run of KE.

3.3. OPAQUE Instantiations

We present several instantiations of OPAQUE using DH-OPRF as the OPRF and different KE protocols. For the sake of concreteness and performance we focus on KE protocols consisting of two or three messages, denoted K1, K2, K3, and such that K1 and K2 include DH values sent by user and server, respectively. These DH values will ensure forward secrecy.

Generic OPAQUE with 3-message KE:

- C to S: Uid, a=H'(PwdU)^r, KE1
- S to C: b=a^k, EnvU, KE2
- C to S: KE3

Key derivation and other details of the protocol are fully specified by the KE scheme.

We provide two instantiations of OPAQUE (with HMQV and SIGMA-I) next and discuss integration with TLS in Section 4).

3.3.1. Instantiation with HMQV

The integration of OPAQUE with HMQV [HMQV] leads to the most efficient instantiation of OPAQUE. It results in a full aAPKE protocol with implicit authentication in just two messages (this inludes the DH-OPRF messages) and with explicit mutual authentication in three. Performance is close to optimal due to the negligible cost of authentication in HMQV: Just 1/6 of an exponentiation for each party over the cost of a regular DH exchange. The private and public keys of the parties are Diffie-Hellman keys, namely, PubU=g^PrivU and PubS=g^PrivS. The HMQV exchange can be represented schematically as follows:

- KE1 = g^x
- KE2 = g^y, Mac(Km1; g^x, g^y)
- KE3 = Mac(Km2; g^y, g^x)

The third message can be removed (as well as the server's Mac) if one is to provide implicit authentication only (e.g., if explicit authentication is achieved by the subsequent protocol or application).

Keys in HMQV, namely, MAC keys Km1, Km2 and session/traffic keys are derived from a common key K computed as follows:

C computes K = H((g^y * PubS^e)^{x + d*PrivU))

S computes K = H((g^x * PubU^d)^{y + e*PrivS))

where d = H(g^x, IdS) and e = H(g^y, IdU), and Idu, IdS represent the identities of user and server. The computation of K involves a single multi-exponentiation whose cost is only 17% more than a regular exponentiation.

This is a minimal skeleton. A fully-specified protocol will include additional details and a careful key derivation scheme. In particular, the Mac computation will cover the whole preceding transcript. In addition, the parties will check group membership for g^x, g^y or use co-factor computation [I-D.irtf-cfrg-spake2] (the check for PubU and PubS can be done only once at user registration).

Note (HMQV patent): IBM has a patent that covers HMQV. While the author does not speak in the name of IBM or with any legal authority, he has reason to believe that if there will be a serious interest in standardizing OPAQUE with HMQV, the patent may not be an impediment.

3.3.2. Instantiation with SIGMA-I

We show how OPAQUE can be built around the 3-message SIGMA-I protocol [SIGMA]. This example is significant as it shows integration with a signature-based KE protocol and because TLS 1.3 follows the design of SIGMA-I hence the example helps understanding the proposed integration of OPAQUE with TLS in Section 4).

SIGMA-I can be represented schematically as follows:

- KE1 = g^x
- KE2 = g^y, Sig(PrivS; g^x, g^y), Mac(Km1; IdS)
- KE3 = Sig(PrivU; g^y, g^x), Mac(Km2; IdU)

In this case, the private keys of both users and servers are signature keys. Key derivation is based on the DH value g^xy.

As before, this is only a skeleton to illustrate the protocol. Full details need to be filled in for a full specification.

3.4. Hardening OPAQUE via user iterations

As noted in Section 2.1 one can add further security to OPAQUE by applying an iterated hash on top of the regular DH-OPRF. For this one changes the computation of RwdU by the user (in the password registration stage and in each online run of OPAQUE) as follows. The user computes DH-OPRF on its password (namely, the value F(kU; PwdU) = H(PwdU, v, (H'(PwdU))^kU)) in interaction with the server using the regular procedure from Section 2. Then it computes RwdU by applying n iterations of a hardening password hash function (see Section 2.1) to F(kU; PwdU). The iteration count n is set at the time of password registration and can be stored at the server and communicated to the user during OPAQUE executions together with the second OPRF message.

4. Integrating OPAQUE with TLS 1.3

Note: This section is intended as a basis for discussion on ways to integrate OPAQUE with TLS (particularly TLS 1.3). Precise protocol details are left for a future specification.

As stated in the introduction, the typical password-over-TLS mechanism for password authentication suffers from significant weaknesses due to the essential reliance of the protocol on PKI and the exposure of passwords to the server (and other observers) upon TLS decryption. Here we propose integrating OPAQUE with TLS in order to remove these vulnerabilities while at the same time armoring TLS itself against PKI failures. Such integration also benefits OPAQUE by leveraging the standardized negotiation and record-layer security of TLS. Furthermore, TLS can offer an initial PKI-authenticated channel to protect the privacy of account information such as user name transmitted between client and server.

If one is willing to forgo protection of user account information transmitted between user and server, integrating OPAQUE with TLS RELATIVELY 1.3 is straightforward and follows essentially the same approach as with SIGMA-I in Section 3.3.2. Specifically, one reuses the Diffie-Hellman exchange from TLS and uses the user's private key PrivU retrieved from the server as a signature key for TLS client authentication. The integrated protocol will have as its first message the TLS's Client Hello augmented with user account information and the DH-OPRF first message (the value a). The server's response includes the regular TLS 1.3 second flight augmented with the second OPRF message which includes the values b, vU and EnvU. For its TLS signature, the server uses the private key PrivS whose corresponding public key PubS is authenticated as part of the user envelope EnvU (there is no need to send a regular TLS certificate in this case). Finally, the third flight consists of the standard client Finish message with client authentication where the

client's signature is produced with the user's private key PrivU retrieved from EnvU and verified by the server with public key PubU.

The above scheme is depicted in Figure 1 where the sign + indicates fields added by OPAQUE; in particular, DH-OPRF1 and DH-OPRF2 denote the two DH-OPRF messages. Other messages in the figure are the same as in TLS 1.3. Notation {...} indicates encryption under handshake keys. Note that ServerSignature and ClientSignature are performed with the private keys defined by OPAQUE and they replace signatures by traditional TLS certificates.

```
    Client                                                  Server

    ClientHello
    key_share
    + userid + DH-OPRF1      -------->
                                                       ServerHello
                                                         key_share
                                              {+ DH-OPRF2 + EnvU}
                                              {+ ServerSignature}
                             <--------           {ServerFinished}

    {+ ClientSignature}
    {ClientFinished}         -------->
```

Figure 1: Integration of OPAQUE in TLS 1.3 (no userid confidentiality)

Adding protection of user's account information is simple using TLS 1.3 pre-shared/resumption mechanisms where the account information appended to the first handshake message would be encrypted under the pre-shared key. The rest of the protocol follows the above description.

When a resumable session or pre-shared key between the client and the server do not exist, user account protection requires a server certificate. In this case, the TLS 1.3 handshake is augmented with the two OPAQUE messages interleaved between the second and third flight of the regular TLS handshake. That is, the protocol consists of five flights as follows: (i) A regular 2-flight 1-RTT handshake to produce handshake traffic keys authenticated by the server's TLS certificate; (ii) two messages that include user identification information, the DH-OPRF messages exchanged between client and server, and the retrieved EnvU, all encrypted under the handshake traffic keys (thus providing privacy to user account information); (iii) the TLS 1.3 client authentication flight where client

authentication uses the user's private signature key PrivU retrieved from the server in step (ii).

Note that server authentication in (i) uses TLS certificates hence user privacy (but not user authentication) is dependent on PKI. In cases where PKI authentication for the server is deemed acceptable then there is no need for further server authentication. However, if one wants to enforce server authentication without reliance on PKI, then the server needs to authenticate using the private key PrivS whose corresponding public key PubS is sent to the user as part of EnvU. There are two options: If PubS is the same as the public key the server used in the 1-RTT authentication (step (i)) then there is no need for further authentication. In this case, U gets assurance from the authenticated EnvU, not from the PKI certificates. Otherwise, the server needs to send a signature under PrivS that is piggybacked to the second OPAQUE message in (ii). In this case the signature would cover the running transcript hash as is standard in TLS 1.3. The client signature in the last message also covers the transcript hash including the regular handshake and OPAQUE messages.

The above scheme is depicted in Figure 2. Please refer to the text before Figure 1 describing notation. Note the asterisk in the ServerSignature message. This indicates that this message is optional as it is used only if the server's key PubS in OPAQUE is different than the one in the server's certificate (transmitted in the second protocol flight).

```
     Client                                                Server

     ClientHello
     key_share                --------->
                                                      ServerHello
                                                        key_share
                                                    {Certificate}
                                              {CertificateVerify}
                              <--------           {ServerFinished}
```

```
        {+ userid + DH-OPRF1}   -------->

                                               {+ DH-OPRF2 + EnvU}
                                <--------     {+ ServerSignature*}

        {ClientSignature}
        {ClientFinished}        -------->
```

Figure 2: Integration of OPAQUE in TLS 1.3 (with userid confidentiality)

We note that the above approaches for integarion of OPAQUE with TLS can benefit from the post-handshake client authentication mechanism of TLS 1.3 and the exported authenticators from [I-D.ietf-tls-exported-authenticator]. Also, formatting of messages and negotiation information suggested in [I-D.barnes-tls-pake] can be used in the OPAQUE setting.

5. User enumeration

User enumeration refers to attacks where the attacker tries to learn whether a given user identity is registered with a server. Preventing such attack requires the server to act with unknown user identities in a way that is indistinguishable from its behavior with existing users. Supporting such defense in OPAQUE requires a modification of the protocol. Note that the server's response to an existing user identity includes two values: a^kU and EnvU. So for a non-existing user these two values need to be sent too. Moreover, the response needs to be the same each time that the same user identity and value a are sent to the server. To achieve this a server can choose the OPRF key kU for a (valid or fake) user "UId" as kU=f(MK; UId) where f is a regular PRF and MK is a server's global key.

The above does not change the protocol as it is a matter of implementation. However, dealing with EnvU for unknown users requires the following change in OPAQUE. In addition to storing EnvU during password registration, the server will also store a value EEK (for EnvU Encryption Key) derived from RwdU by the user. During login, instead of sending EnvU, the server will send a fresh randomized encryption of EnvU under key EEK which the user can decrypt to obtain EnvU after computing RwdU via the OPRF (the rest is the same as before). Since different encryptions of EnvU by the server are independently randomized, the server can simulate such encryption for an unexisting user by encrypting a string of, say, all zeros (or simply sending a random string of the ciphertext's length if the ciphertexts themselves are pseudorandom as in the case of counter mode). Note that both the EEK and the key used to generate EnvU need to be derived from RwdU via a KDF.

[Question: How significant is the user enumeration issue? Should we define OPAQUE as above with built-in defense against enumeration?]

6. Security considerations

This is an early draft presenting the OPAQUE concept and its potential instantiations. More information on implementation and security considerations will be provided in future drafts. We note that the security of OPAQUE is formally proved in [OPAQUE] under a

strong model of aPAKE security assuming the security of the OPRF function and of the underlying key-exchange protocol. In turn, the security of DH-OPRF is proven in the random oracle model under the One-More Diffie-Hellman assumption.

While one can expect the practical security of the OPRF function (namely, the hardness of computing the function without knowing the key) to be in the order of computing discrete logarithms or solving Diffie-Hellman, Brown and Gallant [BG04] and Cheon [Cheon06] show an attack that slightly improves on generic attacks. For the case that q-1 or q+1, where q is the order of the group G, has a t-bit divisor, they show an attack that calls the OPRF on 2^t chosen inputs and reduces security by t/2 bits, i.e., it can find the OPRF key in time 2^{p/2-t/2} and 2^{p/2-t/2} memory. For typical curves, the attack requires an infeasible number of calls and/or results in insignificant security loss [*]. Moreover, in the OPAQUE application, attempting such attacks is completely impractical as the number of calls to the function translates to an equal number of failed authentication attempts by a _single_ user (e.g., one would need a billion impersonation attempts to reduce security by 15 bits and a trillion to reduce it by 20 bits - and most curves will not even allow for such attacks in the first place).

[*] Some examples (courtesy of Dan Brown): For P-384 2^90 calls reduce security from 192 to 147 bits; for NIST P-256 the options are 6-bit reduction with 2153 OPRF calls, about 14 bit reduction with 187 million calls and 20 bits with a trillion calls. For Curve25519, attacks are completely infeasible (require over 2^100 calls) but its twist form allows an attack with 25759 calls that reduces security by 7 bits and one with 117223 calls that reduces security by 8.4 bits.

7. References

7.1. Normative References

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

7.2. Informative References

[Boyen09] Boyen, X., "HPAKE: Password authentication secure against cross-site user impersonation", Cryptology and Network Security (CANS) , 2009.

[BG04] Brown, D. and R. Galant, "The static Diffie-Hellman problem", http://eprint.iacr.org/2004/306 , 2004.

[Canetti01]
Canetti, R., "Universally composable security: A new paradigm for cryptographic protocols", IEEE Symposium on Foundations of Computer Science (FOCS) , 2001.

[Cheon06] Cheon, J., "Security analysis of the strong Diffie-Hellman problem", Euroctypt 2006 , 2006.

[FK00] Ford, W. and B. Kaliski, Jr, "Server-assisted generation of a strong secret from a password", WETICE , 2000.

[GMR06] Gentry, C., MacKenzie, P., and . Z, Ramzan, "A method for making password-based key exchange resilient to server compromise", CRYPTO , 2006.

[I-D.ietf-tls-exported-authenticator]
Sullivan, N., "Exported Authenticators in TLS", draft-ietf-tls-exported-authenticator-07 (work in progress), June 2018.

[I-D.barnes-tls-pake]
Barnes, R. and O. Friel, "Usage of SPAKE with TLS 1.3", draft-barnes-tls-pake-02 (work in progress), June 2018.

[I-D.irtf-cfrg-argon2]
Biryukov, A., Dinu, D., Khovratovich, D., and S. Josefsson, "The memory-hard Argon2 password hash and proof-of-work function", draft-irtf-cfrg-argon2-03 (work in progress), August 2017.

[I-D.irtf-cfrg-spake2]

Ladd, W. and B. Kaduk, "SPAKE2, a PAKE", draft-irtf-cfrg-spake2-05 (work in progress), February 2018.

[I-D.ietf-tls-tls13]
Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", draft-ietf-tls-tls13-28 (work in progress), March 2018.

[I-D.irtf-cfrg-hash-to-curve]
Scott, S., Sullivan, N., and C. Wood, "Hashing to Elliptic Curves", draft-irtf-cfrg-hash-to-curve-01 (work in progress), July 2018.

[OPAQUE] Jarecki, S., Krawczyk, H., and J. Xu, "OPAQUE: An Asymmetric PAKE Protocol Secure Against Pre-Computation Attacks", Eurocrypt , 2018.

[JKKX16] Jarecki, S., Kiayias, A., Krawczyk, H., and J. Xu, "Highly-efficient and composable password-protected secret sharing (or: how to protect your bitcoin wallet online)", IEEE European Symposium on Security and Privacy , 2016.

[SIGMA] Krawczyk, H., "SIGMA: The SIGn-and-MAc approach to authenticated Diffie-Hellman and its use in the IKE protocols", CRYPTO , 2003.

[HMQV] Krawczyk, H., "HMQV: A high-performance secure Diffie-Hellman protocol", CRYPTO , 2005.

[RFC2945] Wu, T., "The SRP Authentication and Key Exchange System", RFC 2945, DOI 10.17487/RFC2945, September 2000, <https://www.rfc-editor.org/info/rfc2945>.

[RFC6628] Shin, S. and K. Kobara, "Efficient Augmented Password-Only Authentication and Key Exchange for IKEv2", RFC 6628, DOI 10.17487/RFC6628, June 2012, <https://www.rfc-editor.org/info/rfc6628>.

[RFC7914] Percival, C. and S. Josefsson, "The scrypt Password-Based Key Derivation Function", RFC 7914, DOI 10.17487/RFC7914, August 2016, <https://www.rfc-editor.org/info/rfc7914>.

Author's Address

Hugo Krawczyk
IBM Research

Email: hugo@ee.technion,ac.il