

**The OPAQUE Asymmetric PAKE Protocol**  
**draft-krawczyk-cfrg-opaque-01**

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 it 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.

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 July 4, 2019.

## Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to [BCP 78](http://trustee.ietf.org/license-info) 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. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## 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 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, protocols discussed recently as candidates for standardization (e.g., SPAKE2+ [[I-D.irtf-cfrg-spake2](#)] and AugPAKE [[RFC6628](#)]) do not enjoy a proof of security, not even in a weak model. The same holds for the SRP protocol [[RFC2945](#)]. VTBPEKE is analyzed in [[VTBPEKE](#)] in a weak model that allows for pre-computation attacks and, as in all the above cases (except OPAQUE), does not accommodate 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 resistance to 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 (or other hardening schemes) and offloading these operations 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. We note that KCI is needed to ensure a crucial property of OPAQUE: even upon compromise of the server, the attacker cannot impersonate the user to the server without first running an exhaustive dictionary attack.

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 [RFC8446] 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 an unambiguous concatenation of strings (details of encodings are left for a future, more detailed specification).

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.

[Author note: It would be good to align this OPRF specification with [\[I-D.sullivan-cfrg-voprf\]](#) and its defined instantiation suites, but





it will require extending that specification to accommodate non-verifiable OPRFs.]

Parameters: Hash function  $H$  (e.g., a SHA2 or SHA3 function), a cyclic group  $G$  of prime order  $q$ , 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  $\alpha=H'(x)*g^r$  to  $S$
- o  $S$ : upon receiving a value  $\alpha$ , respond with  $v=g^k$  and  $\beta=\alpha^k$
- o  $U$ : upon receiving values  $\beta$  and  $v$ , set the PRF output to  $H(x, v, \beta*v^{-r})$

All received values ( $\alpha$ ,  $\beta$ ,  $v$ ) are checked to be elements in  $G$  other than the identity. 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  $\alpha=(H'(x))^r$  in the first message and set the function output to  $H(x,v,\beta^{1/r})$  upon receiving  $S$ 's response. However, note that the multiplicative blinding above allows for a more efficient implementation 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 use a fixed-base optimized computation of the exponentiation  $v^{-r}$ . In any case, it is up to the user to decide on what form of blinding it uses; the server's response is the same in both cases.

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](#)] and references therein.



### **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 would re-define DH-OPRF as  $F(k;x) = I^n( H(x, v, H'(x)^k) )$  where  $I$  is an appropriately chosen function and the symbol  $I^n$  denotes  $n$  iterations of function  $I$ . More generally, any form of hardened password-based key derivation can be used, e.g., PBKDF2 [RFC2898], Argon 2 [I-D.irtf-cfrg-argon2], scrypt [RFC7914]. As we will see, in OPAQUE it is the user who runs this key derivation. The iteration value  $n$  or any other parameter required by these functions can be set as public constants or can be set at time of password registration and later communicated by the server as part of its OPAQUE message. (Note; these hardened KDFs often require a salt value as input; for OPAQUE this value can be set to a constant, e.g., all zeros.)

## **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. In [Section 3.4](#) we augment the protocol with user-side hardening to increase resistance to offline attacks in case of server compromise.

### **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, PKI-based, 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  $k_U$  (random and independent for each user  $U$ ) and sets  $v_U = g^{k_U}$ ; 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  $\text{OPRF}(k_U; \text{PwdU})$  as defined in [Section 2](#) with only U learning the result, denoted RwdU (mnemonics for "Randomized PwdU").
- o U generates an "envelope" EnvU defined as

$$\text{EnvU} = \text{AuthEnc}(\text{RwdU}; \text{PrivU}, \text{PubU}, \text{PubS})$$

where AuthEnc is an authenticated encryption function with the "key committing" property and is specified in [Section 3.1.1](#) below. In EnvU, all values require authentication and PrivU also requires encryption. PubU can be omitted from EnvU if it can be reconstructed from PrivU but while it will save bits on the wire it will come at some computational cost during client authentication.

- o U sends EnvU and PubU to S and erases PwdU, RwdU and all keys. S stores (EnvU, PubS, PrivS, PubU,  $k_U$ ,  $v_U$ ) in a user-specific record. If PrivS and PubS are used for multiple users, S can store these values separately and omit them from the user's record.

Note (salt). We note that in OPAQUE the OPRF key acts as the secret salt value that ensures the infeasibility of pre-computation attacks. No extra salt value is needed.

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.

### [3.1.1](#). Implementing the EnvU envelop

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. This restricts significantly the AuthEnc methods for use, particularly disallowing GCM. We thus specify the AuthEnc function to be implemented via "encrypt-then-hmac" which provably meets this requirement. That is, EnvU will consist of an encryption of the values PrivU, PubU, PubS (concatenated using an unambiguous encoding), with HMAC computed on



the resultant ciphertext. Any secure encryption can be used but the MAC needs to be collision resistant which HMAC satisfies (with a tag size of at least 256 bits). Note that this requires two separate keys for the encryption and MAC parts which can be derived from a single key using, for example, the HKDF-Expand function from [\[RFC5869\]](#). We note that while encryption of PubU and PubS is not strictly required, only authentication, we have included these values under the encryption for the sake of simplicity. Furthermore, encrypting PubU and PubS helps defending against "user enumeration" as discussed in [Section 5](#).

[Author note: Is there a standardized definition of Encrypt-then-HMAC that one can refer? Here is a description from signal [\[SIGNAL\]](#).]

### **3.2. Online OPAQUE protocol**

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

- o User transmits user/account information to the server so that the server can retrieve the user's record.
- o Server sends EnvU and vU to user.
- o Server and User execute the OPRF protocol as defined in [Section 2](#); User sets RwdU to the result of this computation.
- o User decrypts EnvU using RwdU to obtain PrivU, PubU, PubS.
- o User and server run the specified KE protocol using their respective public and private keys.

Note that the steps preceding the run of KE can be arranged in just two messages (one from the user and a response from the server). Furthermore, 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 EnvU, vU and RwdU (i.e., complete the OPRF protocol) 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 (in 2-message protocols K3 is omitted)





and such that  $K_1$  and  $K_2$  include DH values sent by user and server, respectively. These DH values will ensure forward secrecy.

Generic OPAQUE with 3-message KE:

- o C to S:  $U_{id}$ ,  $\alpha = H'(P_{wdU}) * g^r$ ,  $KE_1$
- o S to C:  $\beta = \alpha^{k_U}$ ,  $v_U$ ,  $Env_U$ ,  $KE_2$
- o C to S:  $KE_3$

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 [\[RFC8446\]](#).

### **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 includes 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,  $Pub_U = g^{Priv_U}$  and  $Pub_S = g^{Priv_S}$ . The HMQV exchange can be represented schematically as follows:

- o  $KE_1 = g^x$
- o  $KE_2 = g^y, \text{Mac}(Km_1; g^x, g^y)$
- o  $KE_3 = \text{Mac}(Km_2; g^y, g^x)$

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  $Km_1$ ,  $Km_2$  and session/traffic keys are derived from a common key  $K$  computed as follows:

C computes  $K = H((g^y * Pub_S^e)^{x + d * Priv_U})$

S computes  $K = H((g^x * Pub_U^d)^{y + e * Priv_S})$



where  $d = H(g^x, IdS)$  and  $e = H(g^y, IdU)$ , and  $IdU$ ,  $IdS$  represent the identities of user and server (typically, a user id and domain name, respectively, and can include the party's public key value). The function  $H$  can be the same as used for the DH-OPRF computation. Using multi-exponentiation optimization, 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, e.g., as in [[I-D.irtf-cfrg-spake2](#)]; in any case the value is rejected if it equals the identity. 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 [[RFC8446](#)].

SIGMA-I can be represented schematically as follows:

- o  $KE1 = g^x$
- o  $KE2 = g^y, \text{Sig}(\text{PrivS}; g^x, g^y), \text{Mac}(Km1; IdS)$
- o  $KE3 = \text{Sig}(\text{PrivU}; g^y, g^x), \text{Mac}(Km2; IdU)$

In this case, the private keys of both users and servers are signature keys. Key derivation is based on the DH value  $g^{xy}$ .  $K3$  cannot be omitted.

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



### **3.4. Hardening OPAQUE via hardened key derivation**

As noted in [Section 2.1](#) one can increase the resistance of OPAQUE to offline attacks in case of server compromise by applying a password-hardening key derivation function (KDF) on top of the regular DH-OPRF when computing RwdU. Specifically, the user computes the OPRF in interaction with the server and inputs its result into the KDF in lieu of the password on which this function acts. The output of this computation is set as the value RwdU. Parameters to the KDF function can be set to public values or set at the time of password registration and stored at the server. In this case, the server communicates these parameters to the user during OPAQUE executions together with the second OPRF message. We note that the salt value typically input into the KDF can be set to a constant, e.g., all zeros.

## **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 standard 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 1.3 is relatively 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 with the DH-OPRF first message (the value alpha). The server's response includes the regular TLS 1.3 second flight augmented with the second OPRF message which includes the values beta, 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 using public key PubU.

The above scheme is depicted in Figure 1 where the sign + indicates fields added by OPAQUE, and DH-OPRF1, 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.

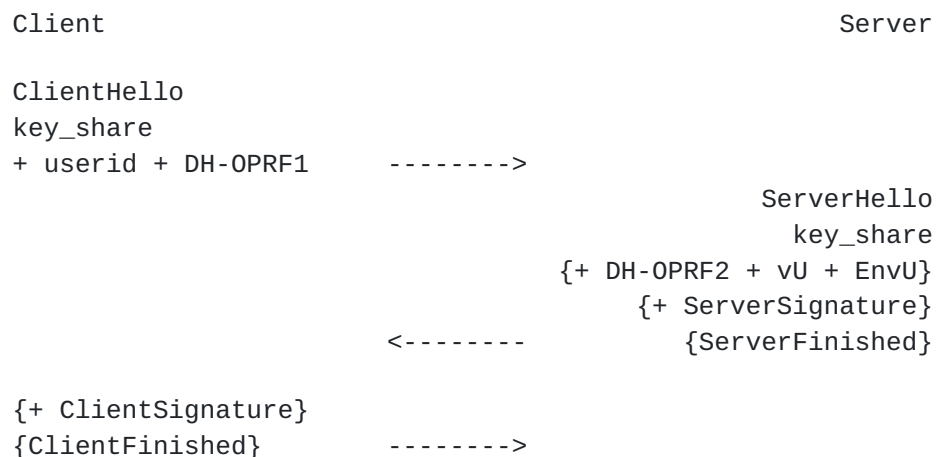


Figure 1: Integration of OPAQUE in TLS 1.3 (no userid confidentiality)

Note that in order to send DH-OPRF1 in the first message, the client needs to know the DH group the server uses for OPRF, or it needs to "guess" it. This issue already appears in TLS 1.3 where the client needs to guess the key\_share group and it should be handled similarly in OPAQUE (e.g., the client may try one or more groups in its first message).

Protection of user's account information can be added through TLS 1.3 pre-shared/resumption mechanisms where the account information appended to the ClientHello message would be encrypted under the pre-shared key.

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 vU and 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 account 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 (only) 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).



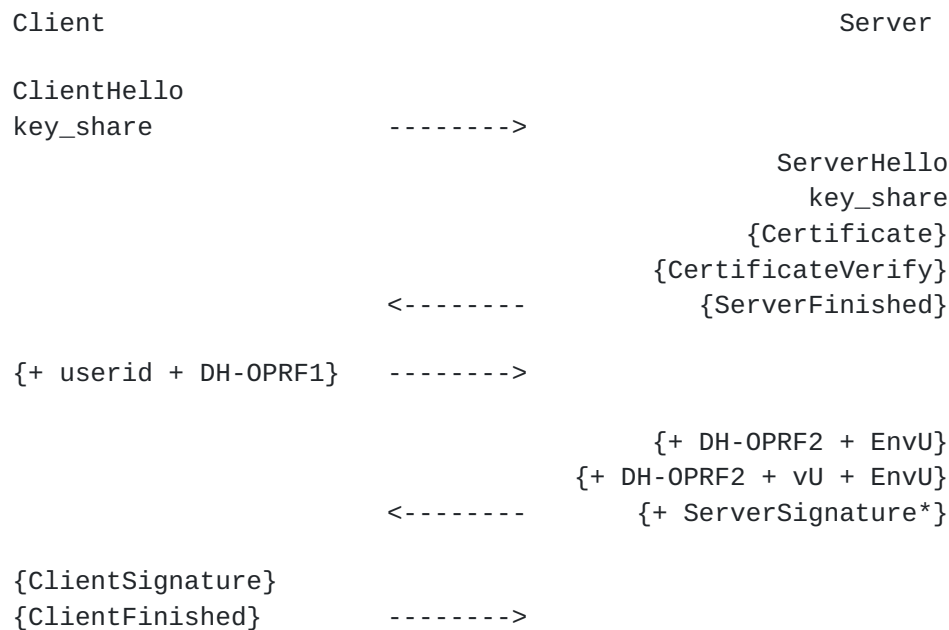


Figure 2: Integration of OPAQUE in TLS 1.3 (with userid confidentiality)

We note that the above approaches for integration of OPAQUE with TLS may 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. Here we suggest a way to implement such defense, namely, a way for simulating the values  $\beta$ ,  $vU$  and  $EnvU$  for non-existing users. Note that if the same pair of user identity  $UId$  and value  $\alpha$  is received twice by the server, the response needs to be the same in both cases (since this would be the case for real users). For this, the server  $S$  will have two keys  $MK$ ,  $MK'$  for a PRF  $f$  (this refers to a regular PRF such as HMAC or CMAC). Upon receiving a pair of user identity  $UId$  and value  $\alpha$  for a non-existing user  $UId$ ,  $S$  computes  $KU=f(MK; UId)$  and  $KU'=f(MK'; UId)$  and responds with three values  $\beta=\alpha^{KU}$ ,  $vU=g^{KU}$  and  $EnvU$  where the latter is computed as the AuthEnc function with key  $KU'$  applied to the all-zero string (of the length of a regular  $EnvU$  plaintext). Care needs to be taken to avoid side channel leakage (e.g., timing) from helping



differentiate these operations from a regular server response. The above requires changes to the server-side implementation but not to the protocol itself or the client side.

There is one form of leakage that the above allows and whose prevention would require a change in OPAQUE. Note that an attacker that tests a `UId` (and same `alpha`) twice and receives different responses can conclude that either the user registered with the service between these two activations or that the user was registered before but changed its password in between the activations (assuming the server changes `KU` at the time of a password change). In any case, this indicates that `UId` is a registered user at the time of the second activation. To conceal this information, `S` can implement the derivation of `KU` as  $KU = f(MK; UId)$  also for registered users. Hiding changes in `EnvU`, however, requires a change in the protocol. Instead of sending `EnvU` as is, we specify that `S` sends an encryption of `EnvU` under a key that the user derives from the OPRF result (similarly to `RwdU`) and that `S` stores during password registration. During login, the user will derive this key from the OPRF result, use it to decrypt `EnvU`, and continue with the regular protocol. If `S` uses a randomized encryption, the encrypted `EnvU` will look each time as a fresh random string, hence `S` can simulate the encrypted `EnvU` also for non-existing users.

[Author note: How significant is the user enumeration issue? The first case above does not change the protocol so its implementation is a server's decision. The second case, however, requires a change in OPAQUE so it would be important to have feedback on whether this second order attack justifies such change.]

## 6. Security considerations

This is an early draft presenting the OPAQUE concept and its potential instantiations. More precise details 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 [JKKX16].

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  $2^{\{q/2-t/2\}}$  and  $2^{\{q/2-t/2\}}$  memory. For typical curves, the attack requires an infeasible number of calls and/or results in insignificant security loss [\*]. Moreover, in the OPAQUE application, these attacks are completely impractical as the number of calls to the function translates to an equal number of failed authentication attempts by a `_single_` user. For example, 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 (note that this theoretical loss of security is with respect to computing discrete logarithms, not in reducing the password strength).

[\*] 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.

Note on user authentication vs. authenticated key exchange. OPAQUE provides PAKE (password-based authenticated key exchange) functionality in the client-server setting. While in the case of user identification, focus is often on the authentication part, we stress that the key exchange element is not less crucial. Indeed, in most cases user authentication is performed to enforce some policy, and the key exchange part is essential for binding this enforcement to the authentication step. Skipping the key exchange part is analogous to carefully checking visitor credentials at the door but then leaving the door open after a first visitor is allowed to enter.

## **7. Acknowledgments**

The OPAQUE protocol is the product of joint work of the author with Stas Jarecki and Jiayu Xu, and builds on a line of work initiated in the seminal paper by Ford and Kaliski [FK00]. This draft has benefited from comments by multiple people. Special thanks to Dan Brown, Eric Crockett, Fredrik Kuivinen, Jason Resch, Nick Sullivan.

## **8. References**

### **8.1. Normative References**

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





## 8.2. 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", Eurocrypt 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.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.



- [I-D.sullivan-cfrg-voprf] Davidson, A., Sullivan, N., and C. Wood, "Verifiable Oblivious Pseudorandom Functions (VOPRFs) in Prime-Order Groups", [draft-sullivan-cfrg-voprf-02](#) (work in progress), October 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.
- [VTBPEKE] Pointcheval, D. and G. Wang, "Verifier-based Two-Basis Password Exponential Key Exchange", AsiaCCS , 2017.
- [SIGNAL] "Signal recommended cryptographic algorithms", <https://signal.org/docs/specifications/doubleratchet/#recommended-cryptographic-algorithms> , 2016.
- [RFC2898] Kaliski, B., "PKCS #5: Password-Based Cryptography Specification Version 2.0", [RFC 2898](#), DOI 10.17487/RFC2898, September 2000, <<https://www.rfc-editor.org/info/rfc2898>>.
- [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>>.
- [RFC5869] Krawczyk, H. and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", [RFC 5869](#), DOI 10.17487/RFC5869, May 2010, <<https://www.rfc-editor.org/info/rfc5869>>.
- [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>>.
- [RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", [RFC 8446](#), DOI 10.17487/RFC8446, August 2018, <<https://www.rfc-editor.org/info/rfc8446>>.

Author's Address

Hugo Krawczyk  
IBM Research

Email: [hugo@ee.technion.ac.il](mailto:hugo@ee.technion.ac.il)

