Crypto Forum Research Group
Internet-Draft
Intended status: Informational
Expires: November 16, 2020

H. Krawczyk
Algorand Foundation

May 15, 2020

The OPAQUE Asymmetric PAKE Protocol
draft-krawczyk-cfrg-opaque-04

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 reliance on PKI and with security against pre-computation attacks upon server compromise. 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, versatility through modularity, 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 instantiations of OPAQUE and ways of integrating OPAQUE with TLS.

This draft presents a high-level description of OPAQUE highlighting its components and modular design. It also provides the basis for a specification for standardization but a detailed specification ready for implementation is beyond the current scope of this document (which may be expanded in future revisions or done separately).

Status of This Memo

Table of Contents

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 (in particular, storage of plaintext passwords is not an uncommon security incident, even among security-conscious companies).

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 may incur additional protocol messages.

This draft describes OPAQUE, a 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 (UC) 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. This includes protocols that have recent analyses in the UC model such as AuCPace [AuCPace] and SPAKE2+ [SPAKE2plus]. We note that as shown in [OPAQUE], these protocols, and any aPAKE in the model from [GMR06], can be converted into an aPAKE secure against pre-computation attacks at the expense of an additional OPRF execution.

It is worth noting that the currently most deployed (OKI-free) aPAKE is SRP [RFC2945] which is open to pre-computation attacks, is inefficient relative to OPAQUE, and does not have an elliptic-curve version (it works for RSA). OPAQUE is therefore a suitable replacement.

OPAQUE’s design builds on a line of work initiated in the seminal paper of Ford and Kaliski [FK00] and is based on the HPAKE protocol of Xavier Boyen [Boyen09] and the (1,1)-PPSS protocol from Jarecki et al. [JKKX16]. None of these papers considered security against pre-computation attacks or 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 (that also helps against online guessing attacks); 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 and forward secrecy - 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. Another essential requirement from KE protocols for use in OPAQUE is to provide forward secrecy (against active attackers).

This draft presents a high-level description of OPAQUE highlighting its components and modular design. It also provides the basis for a specification for standardization but a detailed specification ready for implementation is beyond the current scope of this document (which may be expanded in future revisions or done separately).

We describe OPAQUE with a specific instantiation of the OPRF component over elliptic curves and with a few KE schemes, including the HMQV [HMQV], 3DH [SIGNAL] and SIGMA [SIGMA] protocols. We also present several strategies for integrating OPAQUE with TLS 1.3 [RFC8446] offering different tradeoffs between simplicity, performance and user privacy. In general, the modularity of OPAQUE’s design makes it easy to integrate with additional key-exchange protocols, e.g., IKEv2.

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, two exponentiations for 3DH, and it is one signature generation and verification in the case of SIGMA and TLS 1.3. These instantiations preserve the number of messages in the underlying KE protocol except in one of the TLS instantiations where user privacy may require 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"). Random choices can be replaced with fresh outputs from a cryptographically strong pseudorandom generator or pseudorandom function.

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

2. DH-OPRF

OPAQUE uses in a fundamental way 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 next, generically.

Parameters: Hash function H (e.g., SHA2 or SHA3 function) with 256-bit output at least, 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]

o DH-OPRF Operation: F(k; x) = H(x, 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)^r to S

o S: upon receiving a value alpha, respond with beta=alpha^k

o U: upon receiving beta set the PRF output to H(x, beta^{1/r})

Received values alpha, beta are checked to be elements in G other than the identity and the receiving party aborts if the check fails (alternatively, co-factor exponentiation can be applied to the received values).

Note (fixed-base blinding): An alternative way of computing DH-OPRF is for U to choose random r in [0..q-1] and send alpha=H'(x)_g^r to S, who responds with beta=alpha^k as well as with the value v=g^k (that S may store together with k). U then sets the OPRF output F(k; x) to H(x, beta_v^{-r}). This reduces the computation at U from two variable-base exponentiations in the above protocol to one fixed-base and one variable-base exponentiation. Moreover, if U stores g^k (e.g., for servers to which it logins frequently), then the computation takes two fixed-base exponentiations (with bases g and g^k). The downside of fixed-base blinding is the need for the server to send g^k which is otherwise not necessary. Applications can choose any of the blinding options as both compute the same function.

We note that prior versions of this document defined the OPRF to include g^k under the hash function H in order to provide security for fixed-base blinding. However, [Blinding] proved recently that fixed-base blinding is secure also without hashing g^k.

2.1. DH-OPRF instantiation and detailed specification

The above description of DH-OPRF is generic and applicable to any cyclic group. Detailed specification for concrete implementations of DH-OPEF can be found in [I-D.irtf-cfrg-voprf] which defines several instantiation suites for DH-OPRF, including the choice of hash-to-curve functions (denoted H' above) as detailed in [I-D.irtf-cfrg-hash-to-curve]. OPAQUE will adopt some of these instantiation suites and their underlying elliptic curves. The latter will determine implementation details for such curves including ways to check curve membership, the suitability of co-factor mechanisms, etc.

2.2. Hardening OPRF via user iterations

Protocol OPAQUE is strengthened against offline dictionary attacks by applying to the output of DH-OPRF a hardening procedure such as via repeated iterations, memory hard operations, etc. This greatly increases the cost of an offline attack upon the compromise of the password file at the server. For this purpose, we define the extended DH-OPRF F* as
F*(k; x) = I^n(H(x, H'(x)^k)) where I is a hardening function and n is a measure of hardness. For example, I can represent the iterative function of PBKDF2 [RFC8018] and n the number of iterations; in the case of memory-hard functions such as Argon2 [I-D.irtf-cfrg-argon2] and scrypt [RFC7914], I is a more involved memory-hard function and n measures cost factors and other parameters.

Parameters to the hardening 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.

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 discussed in the introduction). 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, running on a client machine, 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 executed between a user U (running on a client machine) and a server S. It is assumed the server can identify the user and the client can authenticate the server during this registration phase. This is the only part in OPAQUE that requires an authenticated channel, either physical, out-of-band, PKI-based, etc.

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

- S chooses OPRF key kU (random and independent for each user), chooses its own pair of private-public keys PrivS and PubS for use with protocol KE (S can use the same pair of keys with multiple users), and sends PubS to the client.

- Client and S run the OPRF F(kU; PwdU) as defined in Section 2 with only the client learning the result. The client then applies a hardening function, as described in Section 2.2, to this result obtaining a value denoted RwdU (for "Randomized PwdU"). The parameters of the hardening function can be public and known to client machines or they can be stored by S and communicated to the client during registration and login sessions.

- Client generates an "envelope" EnvU that contains PrivU and PubS protected under RwdU. PrivU is encrypted and authenticated while PubS is authenticated and optionally encrypted. EnvU may also include the user’s public key and parties’ identities.

  EnvU can be thought of as an authenticated encryption scheme with optional authenticated-only data. However, for technical reasons, not all authenticated encryption schemes can be used for building EnvU, therefore we provide a precise specification of the enveloping function in Section 4.

- The client sends EnvU and PubU to S and erases PwdU, RwdU and all keys. S stores (EnvU, PubS, PrivS, PubU, kU) 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 never 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 (limited checks at the server are possible to implement, e.g., detecting repeated passwords).

3.2. Online OPAQUE protocol (Login and key exchange))

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

- o Client transmits user/account information to the server so that the server can retrieve the user's record.

- o Server and client execute the OPRF protocol as defined in Section 2; client sets RwdU to the result of this computation (if this computation includes a hardening function as in Section 2.2, the parameters of this function are either known to the client or communicated by the server).

- o Server sends EnvU to client.

- o Client authenticates/decrypts EnvU using RwdU to obtain PrivU, PubU, PubS. If authentication fails, client aborts.

- o Client 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 client 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 client needs to obtain EnvU 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. Parties' identities

Authenticated key-exchange protocols generate keys that need to be uniquely and verifiably bound to a pair of identities, in the case of OPAQUE a user and a server. Thus, it is essential for the parties to agree on such identities, including an agreed bit representation of these identities as needed, for example, when inputting identities to a key derivation function. When referring to identities IdU and IdS in this document, we refer to such agreed identities. Applications may have different policies about how and when identities are determined. A natural approach is to tie IdU to the identity the server uses to fetch EnvU (hence determined during password registration) and to tie IdS to the server identity used by the client to initiate a password registration or login sessions. IdS and IdU can also be part of EnvU or be tied to the parties' public keys. In principle, it is possible that identities change across different sessions as long as there is a policy that can establish if the identity is acceptable or not to the peer. However, we note that the public keys of both the server and the user must always be those defined at time of password registration.

4. Specification of the EnvU envelope

In Section 3.1, EnvU was defined as an envelope containing the user's private key PrivU and server's public key PubS protected under RwdU. Optionally, EnvU may also contain PubU and identities IdS, IdU. Part of this information, e.g., PrivU, requires secrecy and authentication while other values may only need authentication. A natural way to build EnvU is using authenticated encryption with additional authenticated data. However, as proven in [OPAQUE], the security of OPAQUE requires the authenticated encryption scheme, AuthEnc, used to build EnvU to satisfy the property of "random-key robustness". That is, given a pair of random AuthEnc keys, it should be infeasible to create an authenticated ciphertext that successfully decrypts (i.e., passes authentication) under the two keys. Some natural AuthEnc schemes, including GCM, do not satisfy this property and therefore, here we specify a particular scheme for implementing EnvU that enjoys this property. It is based on counter-mode encryption and HMAC.

We define EnvU on the basis of two fields, AEenv and AOenv, one of which (but not both) can be empty. AEenv contains information that needs to be protected under authenticated encryption while AOenv only requires authentication. Typically, AEenv includes PrivU, and AOenv includes PubS and possibly PubU (PubU may be omitted if not needed for running the user side of the key exchange, or if it is re-computed by the client on the basis of PrivU). On the other hand, some applications may want to hide the public key(s) from eavesdroppers in which case these keys would go under AEenv. As noted below, there is also the possibility of omitting PrivU from EnvU and derive it from RwdU in which case AEenv may be empty. In all cases, EnvU must include the authenticated PubS, either under AEenv or AOenv. Additionally, EnvU may be used to transmit the user and/or server identities (see Section 3.3).

EnvU is built by encrypting AEenv (if not empty), concatenating to it AOenv (if not empty), and computing HMAC on the concatenation (which must never be empty). HMAC must use a hash of length 256 bits or more to ensure collision resistance. For the benefit of interoperability we specify the use of a block cipher (AES256) in counter mode as the encryption function, however, any secure (not necessarily authenticated) encryption scheme can be used for the encryption of AEenv. HMAC can also be replaced but only by a collision resistant MAC (not all MAC functions are collision resistant!)

We start by defining the key derivation function to derive three keys: a HMAC key HMACkey, an AES256 key EncKey and a third key KdKey for applications that choose to process user information beyond the OPAQUE functionality (e.g., additional secrets or credentials). We specify KdKey to be of the same length as HMACkey so it can be used, if needed, with HKDF-Expand.

Let L1, L2, L3 be the lengths in octets of HmacKey, EncKey and KdKey, respectively, where L3=L1. If any one of EncKey or KdKey is omitted, its length is set to 0. We define:

KEYS = HKDF(salt=0, IKM=RwdU, info="EnvU", Length=L1+L2+L3)

and set HmacKey to the most significant L1 bytes of KEYS, EncKey to the next significant L2 bytes, and KdKey to the next L3 bytes. (For AES256 and HMAC-SHA256, the keys are of length 32 bytes each.)

We define EnvU to be the concatenation of E and the authentication tag HMAC(HmacKey; E) where E is the concatenation of AES-CTR(EncKey; AEenv) and AOenv.

Recall that EnvU is computed during password registration and is decrypted by the client during login. Decryption proceeds by deriving HmacKey and EncKey, verifying the HMAC tag, and if this is successful, decrypting E. If HMAC verification fails, the session is aborted.

TBC: More precise specification needed here, such as default order of elements, their encodings, etc.

In this specification, encryption of AEenv uses AES256 in counter mode with key EncKey and an initial counter value (that is part of the ciphertext) defined as the concatenation of a random 8-byte nonce chosen by the encrypting party (i.e., the client during password registration) and an 8-byte representation of 1 (7 zero bytes followed by 0x01). We refer to this initial value as CTRBASE.

For completeness, we specify AES-CTR in Appendix Section 9.

TBD: If an RFC defining this mode exists, we should refer to it instead. The mode is defined in [RFC3686] but in the context of IPsec’s ESP, so having a distilled version as in the Appendix may be worthwhile, particularly as we use a different initial value (the above RFC assumes a given IV which we do not have here).

Note (rationale of CTRBASE): The nonce used in defining CTRBASE is needed, for example, for the case where a user registers the same password repeatedly, choosing a fresh PrivU each time while the value of the server’s OPRF key kU stays fixed. This results in the same encryption key but different plaintexts which requires a changing nonce. Eight bytes are more than enough for this.

Note (using GCM): Can one replace AES-CTR with GCM-AES for encrypting AEenv? Yes, as long as one keeps the HMAC authentication. As said, any secure encryption can be used for encrypting AEenv. However, GCM also produces an authentication tag that is not needed here. As a result, using GCM may tempt someone to drop the HMAC authentication which would be insecure since standalone GCM is not random-key robust. For this reason it may be better not to replace plain AES-CTR with GCM or any other authenticated encryption.

Note (storage/communication efficient authentication-only EnvU): It is possible to dispense with encryption in the construction of EnvU to obtain a shorter EnvU (resulting in less storage at the server and less communication from server to client). The idea is to derive PrivU from RwdU. However, for cases where PrivU is not a random string of a given length, we define a more general procedure. Namely, what’s derived from RwdU is a random seed used as an input to a key generation procedure that generates the pair (PrivU, PubU). In this case, AEenv is empty and AOenv contains PubS. The random key generation seed is defined as HKDF-Expand(KdKey; info="KG seed", L) where L is the required seed length. We note that in this encryption-less scheme, the authentication still needs to be random-key robust which HMAC satisfies.

To further minimize storage space, the server can derive per-user OPRF keys kU from a single global secret key, and it can use the same pair (PrivS,PubS) for all users. In this case, the per-user OPAQUE storage consists of PubU and HMAC(Khmac; Pubs), a total of 64-byte overhead with a 256-bit curve and hash. EnvU communicated to the user is of the same length, consisting of PubS and HMAC(Khmac; Pubs).

5. OPAQUE Instantiations

We present several instantiations of OPAQUE using DH-OPRF and different KE protocols. For the sake of concreteness we focus on KE protocols consisting of three messages, denoted KE1, KE2, KE3, and such that KE1 and KE2 include DH values sent by user and server, respectively, and KE3 provides explicit user authentication. As shown in [OPAQUE], OPAQUE cannot use less than three messages so the 3-message instantiations presented here are optimal in terms of number of messages. On the other hand, there is no impediment of using OPAQUE with protocols with more than 3 messages as in the case of IKEv2 (or the underlying SIGMA-R protocol [SIGMA]).

OPAQUE generic outline with 3-message KE:

- C to S: IdU, alpha=H'(PwdU)^r, KE1
- S to C: beta=alpha^kU, EnvU, KE2
- C to S: KE3

Key derivation and other details of the protocol are specified by the KE scheme. We do note that by the results in [OPAQUE], KE2 and KE3 should include authentication of the OPRF messages (or at least of the value alpha) for binding between the OPRF run and the KE session.

Next, we present three instantiations of OPAQUE - with HMQV, 3DH and SIGMA-I. In Section 6 we discuss integration with TLS 1.3 [RFC8446].

5.1. Instantiation of OPAQUE with HMQV and 3DH

The integration of OPAQUE with HMQV [HMQV] leads to the most efficient instantiation of OPAQUE in terms of exponentiations count. Performance is close to optimal due to the low cost of authentication in HMQV: Just 1/6 of an exponentiation for each party over the cost of a regular DH exchange. However, HMQV is encumbered by an IBM patent, hence we also present OPAQUE with 3DH which only differs in the key derivation function at the cost of an extra exponentiation (and less resilience to the compromise of ephemeral exponents). We note that 3DH serves as a basis for the key-exchange protocol of [SIGNAL].

Importantly, many other protocols follow a similar format with differences mainly in the key derivation function. This includes the Noise family of protocols. Extension may also apply to KEM-based KE protocols as in many post-quantum candidates.

The private and public keys of the parties in these examples are Diffie-Hellman keys, namely, PubU=g^PrivU and PubS=g^PrivS.

Specification/implementation details that are specific to the choice of group G will be adapted from the corresponding standards for different elliptic curves.

PROTOCOL MESSAGES. OPAQUE with HMQV and OPAQUE with 3DH comprises:

o KE1 = OPRF1, nonceU, info1_, IdU_, ePubU

o KE2 = OPRF2, EnvU, nonceS, info2_, ePubS, Einfo2_, Mac(Km3; xcript2),

o KE3 = info3_, Einfo3_, Mac(Km3; xcript3)}

where:

o * denotes optional elements;

o OPRF1, OPRF2 denote the DH-OPRF values alpha, beta sent by user and server, respectively, as defined in Section 2;

o EnvU is the OPAQUE’s envelope stored by the server containing keying information for the client to run the AKE with the server;

o nonceU, nonceS are fresh random nonces chosen by client and server, respectively;

o info1, info2, info3 denote optional application-specific information sent in the clear (e.g., they can include parameter negotiation, parameters for a hardening function, etc.);

o Einfo2, Einfo3 denotes optional application-specific information sent encrypted under keys Ke2, Ke3 defined below;

o IdU is the user’s identity used by the server to fetch the corresponding user record, including EnvU, OPRF key, etc. (it can be omitted from message KE1 if the information is available to the server in some other way);

o IdS, the server’s identity, is not shown explicitly, it can be part of an info field (encrypted or not), part of EnvU, or can be known from other context (see Section 3.3); it is used crucially for key derivation (see below);

o ePubU, ePubS are Diffie-Hellman ephemeral public keys chosen by user and server, respectively;

o xcript2 includes the concatenation of the values OPRF1, nonceU, info1_, IdU_, ePubU, OPRF2, EnvU, nonceS, info2_, ePubS, Einfo2_;

o xscript3 includes the concatenation of all elements in xscript2 followed by info3_, Einfo3_;

Notes:

- o The explicit concatenation of elements under xscript2 and xscript3 can be replaced with hashed values of these elements, or their combinations, using a collision-resistant hash (e.g., as in the transcript-hash of TLS 1.3).

- o The inclusion of the values OPRF1 and OPRF2 under xscript2 is needed for binding the OPRF execution to that of the KE session. On the other hand, including EnvU in xscript2 is not mandatory.

- o The ephemeral keys ePubU, ePubS, can be exchanged prior to the above 3 messages, e.g., when running these protocols under TLS 1.3.

KEY DERIVATION. The above protocol requires MAC keys Km2, Km3, and optional encryption keys Ke2, Ke3, as well as generating a session key SK which is the AKE output for protecting subsequent traffic (or for generating further key material). Key derivation uses HKDF [RFC5869] with a combination of the parties static and ephemeral private-public key pairs and the parties' identities IdU, IdS. See Section 3.3.

SK, Km2, Km3, Ke2, Ke3 = HKDF(salt=0, IKM, info, L)

where L is the sum of lengths of SK, Km2, Km3, Ke2, Ke3, and SK gets the most significant bytes of the key stream, Km2 the next bunch, etc.

Values IKM and info are defined for each protocol:

FOR HMQV: Info="HMQV keys" and IKM = Khmqv | IdU | IdS

where Khmqv is computed:

- by the client as: Khmqv = (ePubS * PubS^b)^{ePrivU + a*PrivU}

- by the server as: Khmqv = (ePubU * PubU^a)^{ePrivS + b*PrivS}

with a = H(ePubU, IdS) and b = H(ePubS, IdU)

FOR 3DH: Info="3DH keys" and IKM = K3dh | IdU | IdS

where K3dh is the concatenation of 3 DH values computed

- by the client as: K3dh = ePubS^ePrivU | PubS^ePrivU | ePubS^PrivU

- by the server as: K3dh = ePubU^ePrivS | PubU^ePrivS | ePubU^PrivS

5.2. Instantiation of OPAQUE with SIGMA-I

We show how OPAQUE is built around the 3-message SIGMA-I protocol [SIGMA]. This is an example of a signature-based protocol and also serves as a basis for integration of OPAQUE with TLS 1.3, as the latter follows the design of SIGMA-I (see Section 6. This specification can be extended to the 4-message SIGMA-R protocol as used in IKEv2.

PROTOCOL MESSAGES. OPAQUE with SIGMA-I comprises:

- KE1 = OPRF1, nonceU, info1_, IdU_, ePubU

- KE2 = OPRF2, EnvU, nonceS, info2_, ePubS, Einfo2_, Sign(PrivS; xcript2-), Mac(Km2; IdU),

- KE3 = info3_, Einfo3_, Sign(PrivU; xcript3-), Mac(Km3; IdS)}

See explanation of fields above. In addition, for the signed material, xscript2- is defined similarly to xscript2, however if xscript2 includes information that identifies the user, such information can be eliminated in xscript2- (this is advised if signing user's identification information by the server is deemed to have adverse privacy consequences). In SIGMA, including the peer's identity under the MAC is necessary and sufficient for security, but including it under the signature is not necessary. Similarly, xscript3- is defined as xcript3 with server identification information removed if so desired.

KEY DERIVATION. Key in SIGMA-I are derived as

SK, Km2, Km3, Ke2, Ke3 = HKDF(salt=0, IKM, info, L)

where L is the sum of lengths of SK, Km2, Km3, Ke2, Ke3, and SK gets the most significant bytes of the stream, Km2 the next bunch, etc.

info = "SIGMA-I keys" and IKM is computed

- by the client as IKM = ePubS^ePrivU

- by the server as IKM = ePubU^ePrivS

6. Integrating OPAQUE with TLS 1.3

This section is intended as a discussion of ways to integrate OPAQUE with TLS 1.3. Precise protocol details are left for a future separate specification. A very preliminary draft is [I-D.sullivan-tls-opaque].

As stated in the introduction, the security of the standard password-over-TLS mechanism for password authentication suffers from its essential reliance on PKI and the exposure of passwords to the server (and possibly others) upon TLS decryption. Integrating OPAQUE with TLS removes these vulnerabilities while at the same time it armors TLS itself against PKI failures. Such integration also benefits OPAQUE by leveraging the standardized negotiation and record-layer security of TLS. Furthermore, TLS offers 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 5.2. 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 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 OPRF1, 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 + OPRF1        -------->
                                                 ServerHello
                                                   key_share
                                         {+    OPRF2 + EnvU}
                                           {ServerSignature}
                        <--------           {ServerFinished}

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

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

Note that in order to send 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. One option that does not add round trips is to use a mechanism similar to the proposed ESNI extension [I-D.ietf-tls-esni] or a semi-static TLS exchange as in [I-D.ietf-tls-semistatic-dh]. Without such extensions, one would run a TLS 1.3 handshake augmented with the two first 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 OPAQUE 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 step (i) uses TLS certificates hence PKI is used for user account privacy but not for user authentication or other purposes. (In some applications, PKI may be trusted also for server authentication in which case server authentication through OPAQUE may be forgone). In OPAQUE the server authenticates 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. 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 described 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 + OPRF1}      -------->

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

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

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.

7. 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 and EnvU for non-existing users. Note that if the same pair of user identity IdU 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 protection against this attack, one would apply the encryption function in the construction of EnvU (Section 4) to all the key material in EnvU, namely, AOenv will be empty. 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 IdU and value alpha for a non-existing user IdU, S computes kU=f(MK; IdU) and kU’=f(MK’; IdU) and responds with values beta=alpha^kU and EnvU, where the latter is computed as follows. RwdU is set to kU’ and AEenv is set 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 IdU (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 IdU 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; IdU) also for registered users. Hiding changes in EnvU, however, requires a change in the protocol. Instead of sending EnvU as is, S would send 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, will 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.

Note that the first case above does not change the protocol so its implementation is a server's decision (the client side is not changed). The second case, requires changes on the client side so it changes OPAQUE itself.
TBC: Should this variant be documented/standardized?

8. 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].

Best practices regarding implementation of cryptographic schemes apply to OPAQUE. Particular care needs to be given to the implementation of the OPRF regarding testing group membership and avoiding timing and other side channel leakage in the hash-to-curve mapping. Drafts [I-D.irtf-cfrg-hash-to-curve] and [I-D.irtf-cfrg-voprf] have detailed instantiation and implementation guidance.

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 a visitor's credential at the door and then leaving the door open for others to enter freely.

This draft complies with the requirements for PAKE protocols set forth in [RFC8125].

9. Appendix A. Counter mode encryption

We define counter mode encryption to be used with EnvU (Section 4). The specification is based on [RFC3686] with a different initial value of CTRBLK. The description refers to AES but it applies to any block cipher (with its corresponding block size).

Let PT be the plaintext to be encrypted and CTRBASE a 128-bit initial value (see Section 4 for the OPAQUE-specific CTRBASE value). Partition PT into 128-bit blocks PT = PT[1] PT[2] ... PT[n] where the final block can be shorter than 128 bits. To compute the ciphertext CT, each block PT[i] is XORed with a block KS[i] of a key stream KS obtained by applying AES to a 128-bit counter CTRBLK initialized to CTRBASE and incremented for each block KS[i]. The last value KS[n] is truncated, if necessary, to the length of PT[n]. The ciphertext CT includes n+1 blocks defined as CT[0]=CTRBASE and CT[i]=PT[i] xor KS[i], for i=1,...,n, with the final block possibly shorter than 128 bits.

The encryption of n plaintext blocks can be summarized as:

```
CT[0] := CTRBASE
CTRBLK := CTRBASE
FOR i := 1 to n-1 DO
  CT[i] := PT[i] XOR AES(CTRBLK)
  CTRBLK := CTRBLK + 1
END
CT[n] := PT[n] XOR TRUNC(AES(CTRBLK))
```

The AES() function performs AES encryption with key EncKey. The TRUNC() function truncates the output of the AES encrypt operation to the same length as the final plaintext block, returning the most significant bits.

Decryption is similar. The decryption of ciphertext CT= CT[0] ... CT[n] summarized as:

```
CTRBLK := C[0]
FOR i := 1 to n-1 DO
  PT[i] := CT[i] XOR AES(CTRBLK)
  CTRBLK := CTRBLK + 1
END
PT[n] := CT[n] XOR TRUNC(AES(CTRBLK))
```

10. Appendix B. Acknowledgments

The OPAQUE protocol and its analysis is joint work of the author with Stas Jarecki and Jiayu Xu. We are indebted to the OPAQUE reviewers during CFRG’s aPAKE selection process, particularly Julia Hesse and Bjorn Tackmann. This draft has benefited from comments by multiple people. Special thanks to Richard Barnes, Dan Brown, Eric Crockett, Paul Grubbs, Fredrik Kuivinen, Kevin Lewi, Payman Mohassel, Jason Resch, Nick Sullivan.

11. References

11.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

11.2. Informative References

[AuCPace] Haase, B. and B. Labrique, "AuCPace: Efficient verifier-based PAKE protocol tailored for the IIoT", http://eprint.iacr.org/2018/286 , 2018.

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

[Blinding] Jarecki, S., Krawczyk, H., and J. Xu, "Multiplicative DH-OPRF and Its Applications to Password Protocols", Manuscript , 2020.

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

[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.

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

[I-D.barnes-tls-pake] Barnes, R. and O. Friel, "Usage of PAKE with TLS 1.3", draft-barnes-tls-pake-04 (work in progress), July 2018.

[I-D.ietf-tls-esni] Rescorla, E., Oku, K., Sullivan, N., and C. Wood, "Encrypted Server Name Indication for TLS 1.3", draft-ietf-tls-esni-06 (work in progress), March 2020.

[I-D.ietf-tls-exported-authenticator] Sullivan, N., "Exported Authenticators in TLS", draft-ietf-tls-exported-authenticator-11 (work in progress), December 2019.

[I-D.ietf-tls-semistatic-dh] Rescorla, E., Sullivan, N., and C. Wood, "Semi-Static Diffie-Hellman Key Establishment for TLS 1.3", draft-ietf-tls-semistatic-dh-01 (work in progress), March 2020.

[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-argon2-10 (work in progress), March 2020.

[I-D.irtf-cfrg-hash-to-curve]
Faz-Hernandez, A., Scott, S., Sullivan, N., Wahby, R., and C. Wood, "Hashing to Elliptic Curves", draft-irtf-cfrg-hash-to-curve-07 (work in progress), April 2020.

[I-D.irtf-cfrg-voprf]
Davidson, A., Sullivan, N., and C. Wood, "Oblivious Pseudorandom Functions (OPRFs) using Prime-Order Groups", draft-irtf-cfrg-voprf-03 (work in progress), March 2020.

[I-D.sullivan-tls-opaque]
Sullivan, N., Krawczyk, H., Friel, O., and R. Barnes, "Usage of OPAQUE with TLS 1.3", draft-sullivan-tls-opaque-00 (work in progress), March 2019.

[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.

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

[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>.

[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>.

[RFC8018] Moriarty, K., Ed., Kaliski, B., and A. Rusch, "PKCS #5: Password-Based Cryptography Specification Version 2.1", RFC 8018, DOI 10.17487/RFC8018, January 2017, <https://www.rfc-editor.org/info/rfc8018>.

[RFC8125] Schmidt, J., "Requirements for Password-Authenticated Key Agreement (PAKE) Schemes", RFC 8125, DOI 10.17487/RFC8125, April 2017, <https://www.rfc-editor.org/info/rfc8125>.

[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>.

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

[SIGNAL] "Signal recommended cryptographic algorithms", https://signal.org/docs/specifications/doubleratchet/#recommended-cryptographic-algorithms , 2016.

[SPAKE2plus] Shoup, V., "Security Analysis of SPAKE2+", http://eprint.iacr.org/2020/313 , 2020.

Author's Address

Hugo Krawczyk
Algorand Foundation

Email: hugokraw@gmail.com