**Augmented Password-Authenticated Key Exchange for Transport Layer
Security (TLS)**
draft-shin-tls-augpake-10

Abstract

   This document describes an efficient augmented password-authenticated
   key exchange (AugPAKE) protocol where a user remembers a low-entropy
   password and its verifier is registered in the intended server. In
   general, the user's password is chosen from a small set of
   dictionary, making the password susceptible to offline dictionary
   attacks. The AugPAKE protocol described here is secure against
   passive attacks, active attacks and offline dictionary attacks (on
   the obtained messages with passive/active attacks), and also provides
   resistance to server compromise (in the context of augmented PAKE
   security). Based on the AugPAKE protocol, this document also
   specifies a new password-only authentication handshake for Transport
   Layer Security (TLS) protocol.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

In the real world, many applications such as web mail, Internet
banking/shopping/trade require secure channels between participating
parties. Such secure channels can be established by using an
authenticated key exchange (AKE) protocol, which allows the involving
parties to authenticate each other and to generate a temporary
session key. The temporary session key is used to protect the
subsequent communications between the parties.

Until now, password-only AKE (called, PAKE) protocols have attracted
much attention because password-only authentication is very
convenient to the users. However, it is not trivial to design a
secure PAKE protocol due to the existence of offline dictionary
attacks on passwords. These attacks are possible since passwords are
chosen from a relatively-small dictionary that allows for an attacker
to perform the exhaustive searches. This problem was brought forth
by Bellovin and Merritt [BM92], and many following works have been
conducted in the literature (see some examples in [IEEEP1363.2]). A
PAKE protocol is said to be secure if the best attack an active
attacker can take is restricted to the on-line dictionary attacks,
which allow to check a guessed password only by interacting with the
honest party.

An augmented PAKE protocol (e.g., [BM93], [RFC2945], [ISO]) provides
extra protection for server compromise in the sense that an attacker,
who obtained a password verifier from a server, cannot impersonate
the corresponding user without performing offline dictionary attacks
on the password verifier. This additional security is known as
"resistance to server compromise". The AugPAKE protocol described in
this document is an augmented PAKE which also achieves measurable
efficiency over some previous works (SRP [RFC2945] and AMP [ISO]).
We believe the following (see [SKI10] for the formal security proof):
1) The AugPAKE protocol is secure against passive attacks, active
attacks and offline dictionary attacks (on the obtained messages with
passive/active attacks), and 2) It provides resistance to server
compromise. At the same time, the AugPAKE protocol has similar
computational efficiency to the plain Diffie-Hellman key exchange
[DH76] that does not provide authentication by itself. Specifically,
the user and the server need to compute 2 and 2.17 modular
exponentiations, respectively, in the AugPAKE protocol. After
excluding pre-computable costs, the user and the server are required
to compute only 1 and 1.17 modular exponentiations, respectively.
Compared with SRP [RFC2945] and AMP [ISO], the AugPAKE protocol is
more efficient 1) than SRP in terms of the user's computational costs
and 2) than AMP in terms of the server's computational costs.

Based on the AugPAKE protocol, this document also specifies a new
password-only authentication handshake for Transport Layer Security
(TLS) protocol [RFC5246].

1.1. Keywords

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

2. AugPAKE Specification

2.1. Underlying Group

The AugPAKE protocol can be implemented over the following group.

o Let p and q be sufficiently large primes such that q is a divisor
   of ((p - 1) / 2) and every factors of ((p - 1) / 2) are also
   primes comparable to q in size. This p is called a "secure"
   prime. We denote by G a multiplicative subgroup of prime order q
   over the field GF(p), the integers modulo p. Let g be a generator
   for the subgroup G so that all the subgroup elements are generated
   by g. The group operation is denoted multiplicatively (in modulo
   p).

By using a secure prime p, the AugPAKE protocol has computational
efficiency gains. Specifically, it does not require the order check
of elements, received from the counterpart party. Note that the
groups, defined in Discrete Logarithm Cryptography [SP800-56A] and
RFC 5114 [RFC5114], are not necessarily the above secure prime
groups.

Alternatively, one can implement the AugPAKE protocol over the
following groups.

o Let p and q be sufficiently large primes such that p = (2 * q) +
   1. This p is called a "safe" prime. We denote by G a
   multiplicative subgroup of prime order q over the field GF(p), the
   integers modulo p. Let g be any element of G other than 1. For
   example, g = h^2 mod p where h is a primitive element. The group
   operation is denoted multiplicatively (in modulo p).

o Let p and q be sufficiently large primes such that q is a divisor
   of ((p - 1) / 2). We denote by G a multiplicative subgroup of
   prime order q over the field GF(p), the integers modulo p. Let g
   be a generator for the subgroup G so that all the subgroup
   elements are generated by g. The group operation is denoted
   multiplicatively (in modulo p). If p is not a "secure" prime, the

AugPAKE protocol MUST perform the order check of received
elements.

2.2. Notation

The AugPAKE protocol is a two-party protocol where a user and a
server authenticate each other and generate a session key. The
following notation is used in this document:

U

The user's identity (e.g., defined in [RFC4282]). It is a string
in $\{0,1\}^*$ where $\{0,1\}^*$ indicates a set of finite binary
strings.

S

The server's identity. It is a string in $\{0,1\}^*$.

b = H(a)

A binary string a is given as input to a secure one-way hash
function H (e.g., SHA-2 family [FIPS180-3]) which produces a
fixed-length output b. The hash function H maps $\{0,1\}^*$ to
$\{0,1\}^k$ where $\{0,1\}^k$ indicates a set of binary strings of length
k and k is a security parameter.

b = H'(a)

A binary string a is given as input to a secure one-way hash
function H' which maps the input a in $\{0,1\}^*$ to the output b in
Z_q^* where Z_q^* is a set of positive integers modulo prime q.

a | b

It denotes a concatenation of binary strings a and b in $\{0,1\}^*$.

0x

A hexadecimal value is shown preceded by "0x".

X * Y mod p

It indicates a multiplication of X and Y modulo prime p.

X = g^x mod p

The g^x indicates a multiplication computation of g by x times.
The resultant value modulo prime p is assigned to X. The
discrete logarithm problem says that it is computationally hard
to compute the discrete logarithm x from X, g and p.

w

The password remembered by the user. This password may be used
as an effective password (instead of itself) in the form of
H'(0x00 | U | S | w).

W

   The password verifier registered in the server. This password
   verifier is computed as follows: W = g^w mod p where the user's
   password w is used itself, or W = g^w' mod p where the effective
   password w' = H'(0x00 | U | S | w) is used.

bn2bin(X)

   It indicates a conversion of a multiple precision integer X to
   the corresponding binary string. If X is an element over GF(p),
   its binary representation MUST have the same bit length as the
   binary representation of prime p.

U -> S: msg

   It indicates a message transmission that the user U sends a
   message msg to the server S.

U:

   It indicates a local computation of user U (without any out-going
   messages).

2.2.1. Password Processing

   The input password MUST be processed according to the rules of the
   [RFC4013] profile of [RFC3454]. The password SHALL be considered a
   "stored string" per [RFC3454] and unassigned code points are
   therefore prohibited. The output SHALL be the binary representation
   of the processed UTF-8 character string. Prohibited output and
   unassigned code points encountered in SASLprep pre-processing SHALL
   cause a failure of pre-processing and the output SHALL NOT be used
   with the AugPAKE protocol.

   The following table shows examples of how various character data is
   transformed by the rules of the [RFC4013] profile.

#	Input	Output	Comments
1	I<U+00AD>X	IX	SOFT HYPHEN mapped to nothing
2	user	user	no transformation
3	USER	USER	case preserved, will not match #2
4	<U+00AA>	a	output is NFKC, input in ISO 8859-1
5	<U+2168>	IX	output is NFKC, will match #1
6	<U+0007>		Error - prohibited character
7	<U+0627><U+0031>		Error - bidirectional check

2.3. Protocol

The AugPAKE protocol consists of two phases: initialization and
actual protocol execution. The initialization phase SHOULD be
finished in a secure manner between the user and the server, and it
is performed all at once. Whenever the user and the server need to
establish a secure channel, they can run the actual protocol
execution through an open network (i.e., the Internet) in which an
active attacker exists.

2.3.1. Initialization

```
U -> S: (U, W)
        The user U computes W = g^w mod p (instead of w, the
        effective password w' may be used), and transmits W to the
        server S.  The W is registered in the server as the password
        verifier of user U.  Of course, user U just remembers the
        password w only.
```

If resistance to server compromise is not necessary, the server can
store w' instead of W. In either case, server S SHOULD NOT store any
plaintext passwords.

As noted above, this phase SHOULD be performed securely and all at
once.

2.3.2. Actual Protocol Execution

The actual protocol execution of the AugPAKE protocol allows the user
and the server to share an authenticated session key through an open
network (see Figure 1).

```
  +----------------+                       +-----------------+
  |     User U     |                       | Server S (U,W)  |
  |                |           (U, X)       |                 |
  |                |---------------------------->|            |
  |                |                       |                 |
  |                |           (S, Y)       |                 |
  |                |<----------------------------|            |
  |                |                       |                 |
  |                |            V_U        |                 |
  |                |---------------------------->|            |
  |                |                       |                 |
  |                |            V_S        |                 |
  |                |<----------------------------|            |
  |                |                       |                 |
  +----------------+                       +-----------------+
```
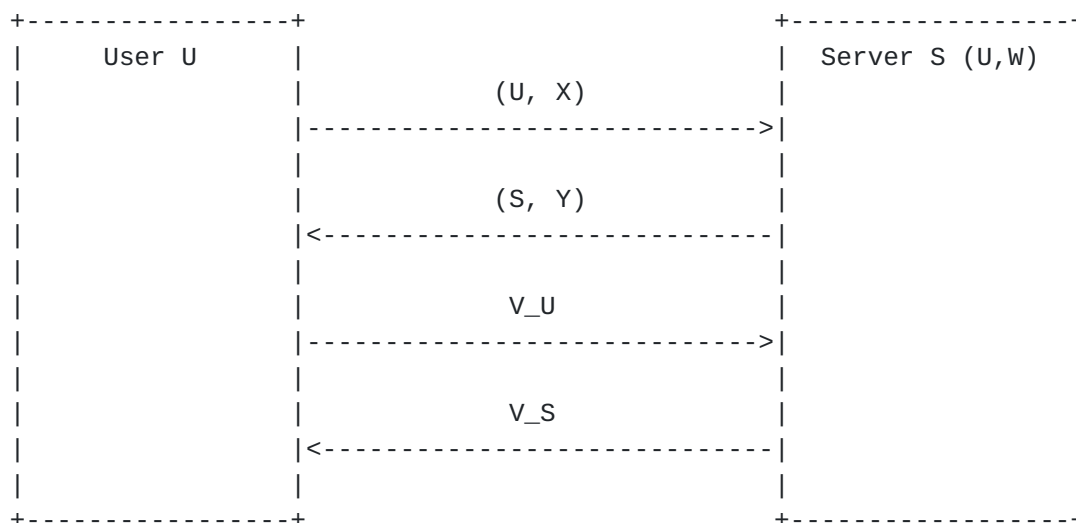
Figure 1: Actual Protocol Execution of AugPAKE

U -> S: (U, X)
      The user U chooses a random element x from Z_q^* and computes
      its Diffie-Hellman public value $X = g^x \bmod p$. The user
      sends the first message (U, X) to the server S.

S -> U: (S, Y)
      If the received X from user U is 0, 1 or -1 (mod p), server S
      MUST terminate the protocol execution. Otherwise, the server
      chooses a random element y from Z_q^* and computes $Y = (X *
      (W^r))^y \bmod p$ where $r = H'(0x01 \mid U \mid S \mid bn2bin(X))$. Note
      that $X^y * g^{(w * r * y)} \bmod p$ can be computed from y and (w
      * r * y) efficiently using Shamir's trick [MOV97]. Then,
      server S sends the second message (S, Y) to the user U.

U -> S: V_U
      If the received Y from server S is 0, 1 or -1 (mod p), user U
      MUST terminate the protocol execution. Otherwise, the user
      computes $K = Y^z \bmod p$ where $z = 1 / (x + (w * r)) \bmod q$ and
      $r = H'(0x01 \mid U \mid S \mid bn2bin(X))$. Also, user U generates an
      authenticator $V_U = H(0x02 \mid U \mid S \mid bn2bin(X) \mid bn2bin(Y) \mid
      bn2bin(K))$. Then, the user sends the third message V_U to
      the server S.

S -> U: V_S
      If the received V_U from user U is not equal to $H(0x02 \mid U \mid
      S \mid bn2bin(X) \mid bn2bin(Y) \mid bn2bin(K))$ where $K = g^y \bmod p$,
      server S MUST terminate the protocol execution. Otherwise,
      the server generates an authenticator $V_S = H(0x03 \mid U \mid S \mid
      bn2bin(X) \mid bn2bin(Y) \mid bn2bin(K))$ and a session key SK =

    H(0x04 | U | S | bn2bin(X) | bn2bin(Y) | bn2bin(K)). Then,
    server S sends the fourth message V_S to the user U.

  U:
    If the received V_S from server S is not equal to H(0x03 |
    U | S | bn2bin(X) | bn2bin(Y) | bn2bin(K)), user U MUST
    terminate the protocol execution. Otherwise, the user
    generates a session key SK = H(0x04 | U | S | bn2bin(X) |
    bn2bin(Y) | bn2bin(K)).

In the actual protocol execution, the sequential order of message
exchanges is very important in order to avoid any possible attacks.
For example, if the server S sends the second message (S, Y) and the
fourth message V_S together, any attacker can easily derive the
correct password w with offline dictionary attacks.

The session key SK, shared only if the user and the server
authenticate each other successfully, MAY be generated by using a key
derivation function (KDF) [SP800-108]. After generating SK, the user
and the server MUST delete all the internal states (e.g., Diffie-
Hellman exponents x and y) from memory.

For the formal proof [SKI10] of the AugPAKE protocol, we need to
change slightly the computation of Y (in the above S -> U: (S, Y))
and K (in the above S -> U: V_S) as follows: Y = (X * (W^r))^y' and K
= g^y' where y' = H'(0x05 | bn2bin(y)).

3. Security Considerations

This section shows why the AugPAKE protocol (i.e., the actual
protocol execution) is secure against passive attacks, active attacks
and offline dictionary attacks, and also provides resistance to
server compromise.

3.1. General Assumptions

o   An attacker is computationally-bounded.

o   Any hash functions, used in the AugPAKE protocol, are secure in
  terms of pre-image resistance (one-wayness), second pre-image
  resistance and collision resistance.

3.2. Security against Passive Attacks

An augmented PAKE protocol is said to be secure against passive
attacks in the sense that an attacker, who eavesdrops the exchanged
messages, cannot compute an authenticated session key (shared between
the honest parties in the protocol).

In the AugPAKE protocol, an attacker can get the messages (U, X), (S, Y), V_U, V_S by eavesdropping, and then wants to compute the session key SK. That is, the attacker's goal is to derive the correct K from the obtained messages X and Y because the hash functions are secure and the only secret in the computation of SK is K = g^y mod p. Note that

X =   g^x mod p and

Y =   (X * (W^r))^y = X^y * W^(r * y) = X^y * (g^y)^t = X^y * K^t

hold where t = w * r mod q. Though t is determined from possible password candidates and X, the only way for the attacker to extract K from X and Y is to compute X^y. However, the probability for the attacker to compute X^y is negligible in the security parameter for the underlying groups since both x and y are random elements chosen from Z_q^*. Therefore, the AugPAKE protocol is secure against passive attacks.

3.3. Security against Active Attacks

An augmented PAKE protocol is said to be secure against active attacks in the sense that an attacker, who completely controls the exchanged messages, cannot compute an authenticated session key (shared with the honest party in the protocol) with the probability better than that of on-line dictionary attacks. In other words, the probability for an active attacker to compute the session key is restricted by the on-line dictioinary attacks where it grows linearly to the number of interactions with the honest party.

In the AugPAKE protocol, the user (resp., the server) computes the session key SK only if the received authenticator V_S (resp., V_U) is valid. There are three cases to be considered in the active attacks.

3.3.1. Impersonation Attacks on User U

When an attacker impersonates the user U, the attacker can compute the same SK (to be shared with the server S) only if the authenticator V_U is valid. For a valid authenticator V_U, the attacker has to compute the correct K from X and Y because the hash functions are secure. In this impersonation attack, the attacker of course knows the discrete logarithm x of X and guesses a password w'' from the password dictionary. So, the probability for the attacker to compute the correct K is bounded by the probability of w = w''. That is, this impersonation attack is restricted by the on-line dictionary attacks where the attacker can try a guessed password communicating with the honest server S. Therefore, the AugPAKE protocol is secure against impersonation attacks on user U.

3.3.2. Impersonation Attacks on Server S

When an attacker impersonates the server S, the attacker can compute
the same SK (to be shared with the user U) only if the authenticator
V_S is valid. For a valid authenticator V_S, the attacker has to
compute the correct K from X and Y because the hash functions are
secure. In this impersonation attack, the attacker chooses a random
element y and guesses a password w'' from the password dictionary so
that

$$Y = (X * (W'^r))^y = X^y * W'^{(r * y)} = X^y * (g^y)^{t'}$$

where $t' = w'' * r \bmod q$. The probability for the attacker to
compute the correct K is bounded by the probability of $w = w''$.
Also, the attacker knows whether the guessed password is equal to w
or not by seeing the received authenticator V_U. However, when w is
not equal to w'', the probability for the attacker to compute the
correct K is negligible in the security parameter for the underlying
groups since the attacker has to guess the discrete logarithm x
(chosen by user U) as well. That is, this impersonation attack is
restricted by the on-line dictionary attacks where the attacker can
try a guessed password communicating with the honest user U.
Therefore, the AugPAKE protocol is secure against impersonation
attacks on server S.

3.3.3. Man-in-the-Middle Attacks

When an attacker performs the man-in-the-middle attack, the attacker
can compute the same SK (to be shared with the user U or the server
S) only if one of the authenticators V_U, V_S is valid. Note that if
the attacker relays the exchanged messages honestly, it corresponds
to the passive attacks. In order to generate a valid authenticator
V_U or V_S, the attacker has to compute the correct K from X and Y
because the hash functions are secure. So, the attacker is in the
same situation as discussed above. Though the attacker can test two
passwords (one with user U and the other with server S), it does not
change the fact that this attack is restricted by the on-line
dictionary attacks where the attacker can try a guessed password
communicating with the honest party. Therefore, the AugPAKE protocol
is also secure against man-in-the-middle attacks.

3.4. Security against Off-line Dictionary Attacks

An augmented PAKE protocol is said to be secure against offline
dictionary attacks in the sense that an attacker, who completely
controls the exchanged messages, cannot reduce the possible password
candidates better than on-line dictionary attacks. Note that, in the
on-line dictionary attacks, an attacker can test one guessed password

by running the protocol execution (i.e., communicating with the
honest party).

As discussed in Section 3.2, an attacker in the passive attacks does
not compute X^y (and the correct K = g^y mod p) from the obtained
messages X, Y. This security analysis also indicates that, even if
the attacker can guess a password, the K is derived independently
from the guessed password. Next, we consider an active attacker
whose main goal is to perform the offline dictionary attacks in the
AugPAKE protocol. As in Section 3.3, the attacker can 1) test one
guessed password by impersonating the user U or the server S, or 2)
test two guessed passwords by impersonating the server S (to the
honest user U) and impersonating the user U (to the honest server S)
in the man-in-the-middle attacks. Whenever the honest party receives
an invalid authenticator, the party terminates the actual protocol
execution without sending any message. In fact, this is important to
prevent an attacker from testing more than one password in the active
attacks. Since passive attacks and active attacks cannot remove the
possible password candidates efficiently than on-line dictionary
attacks, the AugPAKE protocol is secure against offline dictionary
attacks.

3.5. Resistance to Server Compromise

We consider an attacker who has obtained a (user's) password verifier
from a server. In the (augmented) PAKE protocols, there are two
limitations [BJKMRSW00]: 1) the attacker can find out the correct
password from the password verifier with the offline dictionary
attacks because the verifier has the same entropy as the password;
and 2) if the attacker impersonates the server with the password
verifier, this attack is always possible because the attacker has
enough information to simulate the server. An augmented PAKE
protocol is said to provide resistance to server compromise in the
sense that the attacker cannot impersonate the user without
performing offline dictionary attacks on the password verifier.

In order to show resistance to server compromise in the AugPAKE
protocol, we consider an attacker who has obtained the password
verifier W and then tries to impersonate the user U without offline
dictionary attacks on W. As a general attack, the attacker chooses
two random elements c and d from Z_q^*, and computes

X =   (g^c) * (W^d) mod p

and sends the first message (U, X) to the server S. In order to
impersonate user U successfully, the attacker has to compute the
correct K = g^y mod p where y is randomly chosen by server S. After

receiving Y from the server, the attacker's goal is to find out a
value e satisfying $Y^e = K \bmod p$. That is,

$$\log_g (Y^e) = \log_g K \bmod q$$

$$(c + (w * d) + (w * r)) * y * e = y \bmod q$$

$$(c + w * (d + r)) * e = 1 \bmod q$$

where $\log_g K$ indicates the logarithm of K to the base g. Since
there is no offline dictionary attacks on W, the above solution is
that $e = 1 / c \bmod q$ and $d = -r \bmod q$. However, the latter is not
possible since r is determined by X (i.e., r = H'(0x01 | U | S |
bn2bin(X))) and H' is a secure hash function. Therefore, the AugPAKE
protocol provides resistance to server compromise.

3.6. User Privacy

In the AugPAKE protocol, the user's identity U is sent to the server
in clear form. One way to provide user privacy is to use DHE for
encrypting the user's identity.

4. Implementation Consideration

As discussed in Section 3, the AugPAKE protocol is secure against
passive attacks, active attacks and offline dictionary attacks, and
provides resistance to server compromise. However, an attacker in
the on-line dictionary attacks can check whether one password
(guessed from the password dictionary) is correct or not by
interacting with the honest party. Let N be a dictionary size of
passwords. Certainly, the attacker's success probability grows with
the probability of (I / N) where I is the number of interactions with
the honest party. In order to provide a reasonable security margin,
implementation SHOULD take a countermeasure to the on-line dictionary
attacks. For example, it would take about 90 years to test $2^{(25.5)}$
passwords with one minute lock-out for 3 failed password guesses (see
Appendix A in [SP800-63]).

5. AugPAKE for TLS

5.1. Specification of AugPAKE Handshake

The TLS Handshake Protocol [RFC5246], which operates on top of the
TLS record layer, is responsible for negotiating a session and
agreeing upon security parameters (e.g., PreMasterSecret) of the
session state. When protecting subsequent application data, the
agreed security parameters are used by the record layer. The AugPAKE
protocol, described in Section 2, can be easily integrated into the

TLS Handshake Protocol as a password-only mutual authentication by
modifying some messages (see Figure 2). This integrated protocol
preserves the TLS Handshake Protocol structure and its security
guarantees.

```
Client                                                        Server

ClientHello: name, X           -------->
                                                          ServerHello
                                              ServerKeyExchange: Y
                               <--------              ServerHelloDone
ClientKeyExchange
[ChangeCipherSpec]
Finished: V_U                  -------->
                                                   [ChangeCipherSpec]
                               <--------                Finished: V_S

Application Data               <------->              Application Data
```

         Figure 2: Plugging AugPAKE into TLS Handshake Protocol

5.2. Changes from the TLS Handshake Protocol

The changes from the TLS Handshake Protocol are summarized as
follows:

o The X and the TLS client name (it is preferable to use a hashed
   value for privacy protection) are included in the "ClientHello"
   message.

o The Y is included in the "ServerKeyExchange" message.

o The V_U (resp., V_S) is inclued in the TLS client's (resp.,
   server's) "Finished" message.

o The PreMasterSecret in the TLS Handshake Protocol corresponds to K
   of the AugPAKE protocol in Section 2. The leading bytes of K that
   contain all zero bits are stripped before it is used as the
   pre_master_secret.

5.2.1. Changes to Client Hello Message

The extension of the "ClientHello" message will be

```
    enum { AugPAKE (TBD) } ExtensionType;

    struct {
```

```
     opaque name<1..2^8-1>;

     opaque X<1..2^16-1>;

  } Extension;
```

where the extension data field of the AugPAKE extension SHALL contain
a "name" to be used to identify the TLS client and the client's
Diffie-Hellman public value X.

If the TLS server does not find a verifier corresponding to the
"name" in the extension of the "ClientHello" message, the server
SHOULD keep running the protocol by choosing a random element Y and
then rejects the TLS client's "Finished" message with a
bad_record_mac alert. Alternatively, the TLS server MAY terminate
the protocol if a verifier corresponding to the "name" in the
extension of the "ClientHello" message is not found.

5.2.2. Changes to Server Key Exchange Message

The "ServerKeyExchange" message contains the server's computed value
Y (see Section 2). Also, a new value is added to the
KeyExchangeAlgorithm to indicate its use of AugPAKE.

```
  enum { augpake } KeyExchangeAlgorithm;

  struct { opaque Y<1..2^16-1>; } ServerAugPAKEParams;
```

6. Acknowledgement

We would sincerely appreciate comments from Fabrice Gautier, Watson
Ladd and IIari Liusvaara. The AugPAKE protocol described in
Section 2 has also been specified as a password-only authentication
method for IKEv2.

7. IANA Considerations

This document requests IANA to assign a value.

IANA SHALL assign a value for "AugPAKE" from the TLS ExtensionType
Registry defined in [RFC5246] with the method name of "AugPAKE".

8. References

8.1. Normative References

   [FIPS180-3]
              Information Technology Laboratory, , "Secure Hash Standard
              (SHS)", NIST FIPS Publication 180-3, October 2008,
              <http://csrc.nist.gov/publications/fips/fips180-3/
              fips180-3_final.pdf>.

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

   [RFC3454] Hoffman, P. and M. Blanchet, "Preparation of
              Internationalized Strings ("stringprep")", RFC 3454,
              December 2002.

   [RFC4013] Zeilenga, K., "SASLprep: Stringprep Profile for User Names
              and Passwords", RFC 4013, February 2005.

   [RFC4282] Aboba, B., Beadles, M., Arkko, J., and P. Eronen, "The
              Network Access Identifier", RFC 4282, December 2005.

   [RFC5246] Dierks, T., Rescorla, E., , and , "The Transport Layer
              Security (TLS) Protocol Version 1.2", RFC 5246, August
              2008.

   [SP800-108]
              Chen, L., "Recommendation for Key Derivation Using
              Pseudorandom Functions (Revised)", NIST Special
              Publication 800-108, October 2009,
              <http://csrc.nist.gov/publications/nistpubs/800-108/
              sp800-108.pdf>.

8.2. Informative References

   [BJKMRSW00]
              Bellare, M., Jablon, D., Krawczyk, H., MacKenzie, P.,
              Rogaway, P., Swaminathan, R., and T. Wu, "Proposal for
              P1363 Study Group on Password-Based Authenticated-Key-
              Exchange Methods", IEEE P1363.2: Password-Based Public-Key
              Cryptography , Submissions to IEEE P1363.2 , February
              2000, <http://grouper.ieee.org/groups/1363/passwdPK/
              contributions/p1363-pw.pdf>.

   [BM92]   Bellovin, S. and M. Merritt, "Encrypted Key Exchange:
              Password-based Protocols Secure against Dictionary
              Attacks", Proceedings of the IEEE Symposium on Security
              and Privacy , IEEE Computer Society , 1992.

   [BM93]   Bellovin, S. and M. Merritt, "Augmented Encrypted Key
              Exchange: A Password-based Protocol Secure against
              Dictionary Attacks and Password File Compromise",
              Proceedings of the 1st ACM Conference on Computer and
              Communication Security , ACM Press , 1993.

   [DH76]   Diffie, W. and M. Hellman, "New Directions in
              Cryptography", IEEE Transactions on Information Theory
              Volume IT-22, Number 6, 1976.

   [IEEEP1363.2]
              IEEE P1363.2, and , "Password-Based Public-Key
              Cryptography", Submissions to IEEE P1363.2 ,
              <http://grouper.ieee.org/groups/1363/passwdPK/
              submissions.html>.

   [ISO]   ISO/IEC JTC 1/SC 27 11770-4, and , "Information technology
              -- Security techniques -- Key management -- Part 4:
              Mechanisms based on weak secrets", May 2006,
              <http://www.iso.org/iso/iso_catalogue/catalogue_tc/
              catalogue_detail.htm?csnumber=39723>.

   [MOV97]   Menezes, A., Oorschot, P., and S. Vanstone, "Simultaneous
              Multiple Exponentiation", in Handbook of Applied
              Cryptography , CRC Press , 1997.

   [RFC2945] Wu, T. and , "The SRP Authentication and Key Exchange
              System", RFC 2945, September 2000.

   [RFC5114] Lepinski, M. and S. Kent, "Additional Diffie-Hellman
              Groups for Use with IETF Standards", RFC 5114, January
              2008.

   [SK]   Shin, S. and K. Kobara, "Augmented Password-Authenticated
              Key Exchange (AugPAKE)", (work in progress), August 2014,
              <https://datatracker.ietf.org/doc/draft-irtf-cfrg-
              augpake/>.

   [SKI10]   Shin, S., Kobara, K., and H. Imai, "Security Proof of
              AugPAKE", Cryptology ePrint Archive: Report 2010/334,
              June 2010, <http://eprint.iacr.org/2010/334>.

[SP800-56A]
          Barker, E., Johnson, D., and M. Smid, "Recommendation for
          Pair-Wise Key Establishment Schemes Using Discrete
          Logarithm Cryptography (Revised)", NIST Special
          Publication 800-56A, March 2007,
          <http://csrc.nist.gov/publications/nistpubs/800-56A/
          SP800-56A_Revision1_Mar08-2007.pdf>.

[SP800-63]
          Burr, W., Dodson, D., and W. Polk, "Electronic
          Authentication Guideline", NIST Special Publication 800-63
          Version 1.0.2, April 2006,
          <http://csrc.nist.gov/publications/nistpubs/800-63/
          SP800-63V1_0_2.pdf>.

Appendix A. Features of AugPAKE

Below are some features of the AugPAKE protocol.

Security:

   AugPAKE is zero knowledge (password) proof. It is secure against
   passive/active/offline dictionary attacks. It is also resistant
   to server-compromise impersonation attacks.

   AugPAKE provides Perfect Forward Secrecy (PFS) and is secure
   against Denning-Sacco attack.

   Any cryptographically secure Diffie-Hellman groups can be used.

   The formal security proof of AugPAKE can be found at [SKI10].

   AugPAKE can be easily used with strong credentials.

   In the case of server compromise, an attacker has to perform
   offline dictionary attacks while computing modular exponentiation
   with a password candidate.

Intellectual Property:

   AugPAKE was publicly disclosed on Oct. 2008.

   AIST applied for a patent in Japan on July 10, 2008. AIST would
   provide royal-free license of AugPAKE.

   IPR disclosure (see https://datatracker.ietf.org/ipr/2037/)

Miscellaneous:

The user needs to compute only 2 modular exponentiation
computations while the server needs to compute 2.17 modular
exponentiation computations. AugPAKE needs to exchange 2 group
elements and 2 hash values. This is almost the same computation/
communication costs as the plain Diffie-Hellman key exchange. If
we use a large (e.g., 2048/3072-bits) parent group, the hash size
would be relatively small.

AugPAKE has the same performance for any type of secret.

Internationalization of character-based passwords can be
supported.

AugPAKE can be implemented over any ECP (Elliptic Curve Group over
GF[P]), EC2N (Elliptic Curve Group over GF[2^N]), and MODP
(Modular Exponentiation Group) groups. See Appendix 3 of[SK].

AugPAKE has request/response nature.

No Trusted Third Party (TTP) and clock synchronization

No additional primitive (e.g., Full Domain Hash (FDH) and/or ideal
cipher) is needed.

Easy implementation. We already implemented AugPAKE and have been
testing in AIST.

Appendix B. Test Vector of AugPAKE

Here is a test vector of the AugPAKE protocol.

```
p = 0x FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
       4300000000000000000000000000000000000000000000000000000000000000
       0000000000000000000000000000000000000000000000000000000000000000
       0000000000000000000000000000000000000000000000000000000000000000
       0000000000000000000000000000000000000000000000000000000000000000
       0000000000000000000000000000000000000000000000000000000000000000
       0000000000000000000000000000000000000000000000000000000000000000
       0000000000000000000000000000000000000000000000000000000000000000
       0000000000000000000000000000000000000000000000000000000000000000
       0000000000000000000000000000000000000000000000000000000000000000
       0000000000000000000000000000000000000000000000000000000000000000
       0000000000000000000000000000000000000000000000000000000000000000
       000000000000000330A0DFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
       FFFFFFFFFFFFFFFFFDA5193AB

q = 0x FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
       43
```

```
g = 0x F1AC99884ABBBBCC9BAA19BF375607FD14570B3019A03871147032445ADA7F
       A5B8BDC399C1889BBDA197ADB1E3939D55361241F5CD5ED529B0ADD921B274
       44BD2EB698DC962A9F7D202EAB98BC0C8CC950CA13BC6B1E632D0876A4E796
       26FDE85F06A46C9991EB02A6D6096E0DF6BCA2CAA12E838BEC47A7CB4AF2B0
       D94107B9CDBD67327238ECAF84DF292E776AF0F76288B39F9D9E4DDF3A9731
       CC832D70F150A0F29E7A1E193D1D21CBE8A84B56B0A4692CB39D3048086782
       85A23F08F9DB402487746F7E2A19CAF2171E55C76337E359217516213FF3BF
       616F8B20586A8B3168DA444AEA862BB76B9EA2BF8CB84773D29D4EFE511C53
       95F89CB547EFBBAE333E0BDB22DA40CE0B942A59841A12790910CC1332699D
       64BBF667E0DF3791C4E29CEB48E8397D50C72F7765C5A18809E3497F6BD374
       F5D185BBC8F57E36051E11E8DD0C5DD385A9DA442F22598111960CC2B83CBA
       0A1D980745562F6C62DD6D81B7BAEA7650B1E6E57AB9CC4C95EF17256A79B1
       31859E1BAC81FF1E

U =    augpakeuser@aist.go.jp

S =    augpakeserver@aist.go.jp

X = 0x AF261CB6FD99E2E3B4FCDE8A9C538A872EBB54CC82845E8038EA34804DD739
       90689F2191BF6436BFFD13459DC5FF1FBD56734CA6347ADF52AFAB85EC98A0
       4D27C122EBAF1840229C2F74E9AEEB79C27ECAF6C8B88FCD5FB3F9C8F20F8F
       88D45CEBB56D6CAB251DC34D3656DB73E4DF77B71420565D4696DBF8D27023
       36E909F4C33B2F14605BDEB535423F02B779D7AF0CCB4F3705E177C4334F14
       707140218197845C708E6ABC025BCA954AA2FDAA352D284DE41D33F7CCFCFD
       99CAB3CFD34361B6D0551C6D8AEE604316F99758A9C58A96C39AA811E8BA10
       2E4D21D8F5EFC26920B3F95E6C800D8EBE9914B53144B038BE0E0B43D86479
       E3CBE623FD148CC90784548C1C7C522C6C71FD4A095B15CBF3EB74C7759493
       38AC7691A3763C0A3AB97E9E2B439192C49470BF90F292C4A8D336D86780FD
       DC7DC3C2CC04A6E3AB632788D97D4FDDEE5B117DCF76D91CEA1794FAD91CA1
       7C9715D83589A00CBB72DD513F6A30E4DB31ACA8DA5D6ECD0E20307D70D6DD
       0C2D6D3D7962694B5E1B0848

Y = 0x 39033C00E1C30FF1A945DD699C516D3BB194FABBB46ADF54A2ED0820FEE0A4
       DE52128E2126A4393E3040E07D90432B8C19433EC920274E5A7C165DC89AA2
       C6D77E801A0C7676BC71955A52E74775C57C63B97304C73211C2035BD7B6D1
       8844A28AC50DD2DA96B257236A6090CCB08D2006BF57AC69EA14ABC5D71BA0
       6A6BD4093A2EF27C74D5D9189BB2EC865E6321D0DFECDE2D9AC537E8254E98
       C38AE00BE2554F71FE6EFBBDBE8D7038128957E98C99206998B0B4E578EC47
       957205C228FAC57B9A1589B8FF2B134980504F56B8A84809B8FF70EFF67520
       2B255F0724DC0F76F3802D8A42ACC33D349A7FAF249BFBFEB324C3966D2B30
       6093C32A928A8BEB99AC301D20372E95BB8A3E500778B4651EB8A19B162666
       8DDFB77D0DF4C1932F1FE63389F3B1F29AE99F34BC39EF0AD04BC3A6A129DE
       E66E50B6768EDECC529F06FE5F7AD3825E8ECFCB12DB579C40F19D12BF6F60
       4621F60413DAEB77FE48C136518C57D02A2C6BB596EDFA0DACC127C2FD5FE1
       9B72580A722307C3F86C0EB1

V_U = 0x  490C7CE33DCC3EBE8D0406EEB97CA154882DCBBA0A728F3B870263BCA36
          9DB6
```

```
V_S = 0x  D70D2CAA821B9D84E29D75EB5E9B2DB038BA1256ECFC35C553832743A6E
          36F
```

Appendix C. AugPAKE over EC Groups

The AugPAKE protocol can be implemented over any elliptic curve
groups. The following is such an example.

Let p and q be sufficiently large primes, and let m be some positive
integer. An elliptic curve E is defined by one of the following two
curve equations

        $y^2 = x^3 + a * x + b$ over the prime field GF(p) or

        $y^2 + x * y = x^3 + a * x^2 + b$ over the binary field GF(2^m)

together with the point at infinity 0_E where x, y, and two
coefficients a and b are elements of GF(p) or GF(2^m). Let #E be the
number of points on E, and prime q be the order of the desired group.
The cofactor k is the value (#E / q) satisfying k = 2^n * q_1 * q_2
... q_t where n = {0,1,2} and every primes q_i > q for i = 1, 2,
..., t. Optionally, k = 2^n. Also, n can be 3 for good performance
and security. Let G be a generator for a subgroup of q points on E
so that all the subgroup elements are generated by G. The group
operation is denoted additively. For example, (X = [x] * G)
indicates that an addition computation of G by x times and the
resultant value is assigned to X.

By using the above elliptic curve groups, the AugPAKE protocol has
computational efficiency gains. Specifically, it does not require
the order check of elements, received from the counterpart party.

The AugPAKE protocol consists of two phases: initialization and
actual protocol execution. The initialization phase SHOULD be
finished in a secure manner between the user and the server, and it
is performed all at once. Whenever the user and the server need to
establish a secure channel, they can run the actual protocol
execution through an open network (i.e., the Internet) in which an
active attacker exists.

Initialization

   U -> S: (U, W)
           The user U computes W = [w] * G (instead of w, the
           effective password w' may be used), and transmits W to the
           server S. The W is registered in the server as the
           password verifier of user U. Of course, user U just
           remembers the password w only.

Actual Protocol Execution

    U -> S: (U, X)
            The user U chooses a random element x from Z_q^* and
            computes its elliptic curve Diffie-Hellman public value X
            = [x] * G. The user sends the first message (U, X) to the
            server S.

    S -> U: (S, Y)
            If the received X from user U is not a point on E or [2^n]
            * X = 0_E, server S MUST terminate the protocol execution.
            Otherwise, the server chooses a random element y from
            Z_q^* and computes Y = [y] * (X + ([r] * W)) where r =
            H'(0x01 | U | S | bn2bin(X)). Then, server S sends the
            second message (S, Y) to the user U.

    U -> S: V_U
            If the received Y from server S is not a point on E or
            [2^n] * Y = 0_E, user U MUST terminate the protocol
            execution. Otherwise, the user computes K = [z] * Y where
            z = 1 / (x + (w * r)) mod q and r = H'(0x01 | U | S |
            bn2bin(X)). Also, user U generates an authenticator V_U =
            H(0x02 | U | S | bn2bin(X) | bn2bin(Y) | bn2bin(K)).
            Then, the user sends the third message V_U to the server
            S.

    S -> U: V_S
            If the received V_U from user U is not equal to H(0x02 |
            U | S | bn2bin(X) | bn2bin(Y) | bn2bin(K)) where K = [y] *
            G, server S MUST terminate the protocol execution.
            Otherwise, the server generates an authenticator V_S =
            H(0x03 | U | S | bn2bin(X) | bn2bin(Y) | bn2bin(K)) and a
            session key SK = H(0x04 | U | S | bn2bin(X) | bn2bin(Y) |
            bn2bin(K)). Then, server S sends the fourth message V_S
            to the user U.

        U:
            If the received V_S from server S is not equal to H(0x03 |
            U | S | bn2bin(X) | bn2bin(Y) | bn2bin(K)), user U MUST
            terminate the protocol execution. Otherwise, the user
            generates a session key SK = H(0x04 | U | S | bn2bin(X) |
            bn2bin(Y) | bn2bin(K)).

Authors' Addresses

SeongHan Shin
AIST
2-4-7 Aomi, Koto-ku
Tokyo, Tokyo 135-0064
JP

Phone: +81-3-3599-8001
Email: seonghan.shin@aist.go.jp

Kazukuni Kobara
AIST

Email: kobara_conf-ml@aist.go.jp