X3DH

1. Introduction.

This document describes the X3DH (Extended Triple Diffie-Hellman) key agreement protocol. X3DH establishes a shared secret key between two parties who mutually authenticate each other based on public keys.

X3DH provides forward secrecy and cryptographic deniability.

X3DH is designed for asynchronous settings where one user (“Bob”) is offline but has published some information to a server. Another user (“Alice”) wants to use that

information to send encrypted data to Bob, and also establish a shared secret key for future communication.

2. Overview.

2.1. XD3H main parameters.

An application using X3DH must specifies these parameters:

Name	Definition
curve	X25518 or X448
hash	A 256-bits or 512-bits cryptographic hash function
info	An ASCII string identifying the application

An application must additionally define an encoding function $Encode(PK)$ to encode an X25519 or X448 public key PK into a byte sequence. The recommended encoding consists of some single-byte constant to represent the type of curve, followed by little-endian encoding of the u-coordinate as specified in [1].

2.2. Cryptographic notation and functions.

- The concatenation of two byte sequences X and Y is $X||Y$
- Function $DH(PK_1, PK_2)$: a byte sequence which is the output of a Elliptic Curve Diffie-Hellman function, the input of which is a pair of public keys (PK_1, PK_2).

  Depending on the curve parameter, the Elliptic Curve Diffie-Hellman function will either be the X25519 or X488 function defined in [1].

```
X25519(k, u): # or X448(k, u)
              # note that these formulas are slightly different from Montgomery's
              # original paper.  Implementations are free to use any correct
              # formulas
  x_1 = u
  x_2 = 1
  z_2 = 0
  x_3 = u
  z_3 = 1
  swap = 0
  For t = bits-1 down to 0:
    k_t = (k >> t) & 1
    swap ^= k_t
```

```
    # Conditional swap should be of constant time
    (x_2, x_3) = cswap(swap, x_2, x_3)
    (z_2, z_3) = cswap(swap, z_2, z_3)
    swap = k_t

    A = x_2 + z_2
    AA = A^2
    B = x_2 - z_2
    BB = B^2
    E = AA - BB
    C = x_3 + z_3
    D = x_3 - z_3
    DA = D * A
    CB = C * B
    x_3 = (DA + CB)^2
    z_3 = x_1 * (DA - CB)^2
    x_2 = AA * BB
    z_2 = E * (AA + a24 * E)

  # Conditional swap should be of constant time
  (x_2, x_3) = cswap(swap, x_2, x_3)
  (z_2, z_3) = cswap(swap, z_2, z_3)
  Return x_2 * (z_2^(p - 2))
```

The cswap function SHOULD be implemented in constant time (i.e., independent of the swap argument). For example:

```
cswap(swap, x_2, x_3):
  dummy = mask(swap) AND (x_2 XOR x_3) # mask(swap) is the all-1 or all-0
                                       # word of the same length as x_2 and x_3,
                                       # computed, e.g., as mask(swap) = 0 - swap.
  x_2 = x_2 XOR dummy
  x_3 = x_3 XOR dummy
  Return (x_2, x_3)
```

- Function $Sig(PK, M)$: a byte sequence that is an XEdDSA signature on the byte sequence M and verifies with public key PK. PK is created by signing M with PK's private key. The signing and verification functions of XEdDSA are specified in [2].
- Function $KDF(KM)$: a 32 bytes output from the $HKDF$ agorithm [3] from the inputs:
  - HKDF input key material $F \| KM$, where KM is an input byte sequence containing secret key material, and F is a byte sequence containing 32 0xFF

bytes if curve is X25519, and 57 0xFF bytes if curve is X448. F is used for cryptographic domain separation with XEdDSA [2].

- HKDF salt: A zero-filled byte sequence with length equal to the hash output length.
- HKDF info: The info parameter from Section 2.1.

2.3. Roles.

The X3DH protocol involves three parties: Alice, Bob, and a server.

- Alice wants to send Bob some initial data using encryption, and also establish a shared secret key which may be used for bidirectional communication.
- Bob wants to allow parties like Alice to establish a shared key with him and send encrypted data. However, Bob might be offline when Alice attempts to do this. To enable this, Bob has a relationship with some server.
- The server can store messages from Alice to Bob which Bob can later retrieve. The server also lets Bob publish some data which the server will provide to parties like Alice. The amount of trust placed in the server is discussed in Section 4.7.

  The server role might be divided amongst multiple entities, but for simplicity we assume a single server for this document.

2.4. Keyes.

X3DH uses the following elliptic curve public keys:

Name	Definition
IK_A	Alice's identity key
EK_A	Alice's ephemeral key
IK_B	Bob's identity key
SPK_B	Bob's signed prekey
OPK_B	Bob's one-time prekey

Note that all public keys have a corresponding private key. The public keys used within an X3DH protocol run must either all be in X25519 form, or they must all be in X448 form, depending on the curve parameter [1].
Each party has a long-term identity public key (IK_A for Alice, IK_B for Bob). Bob also

has a signed prekey SPK_B, which he will change periodically, and a set of one-time prekeys OPK_B, which are each used in a single X3DH protocol run.

(Prekeys are essentially protocol messages which Bob publishes to the server prior to Alice beginning the protocol run).

During each protocol run, Alice generates a new ephemeral key pair with public key EK_A. After a successful protocol run Alice and Bob will share a 32-byte secret key SK. This key may be used within some post-X3DH secure communication protocol, subject to the security considerations in Section 4.

3. The XD3H protocol.

3.1. Overview.

X3DH has three phases:

1. Bob publishes his identity key and prekeys to a server.
2. Alice fetches a “prekey bundle” from the server, and uses it to send an initial message to Bob.
3. Bob receives and processes Alice’s initial message.
   (Nên thêm 1 hình vẽ ở đây nhưng chưa có thời gian chau chuốt)

3.2. Publishing keys.

Bob publishes a set of elliptic curve public keys to the server, containing:

- Bob’s identity key IKB
- Bob’s signed prekey SPK_B
- Bob’s prekey signature $Sig(IK_B, Encode(SPK_B))$
- A set of Bob’s one-time prekeys (OPK_{B^1}, OPK_{B^2}, ...)

Bob only needs to upload his identity key to the server once. However, Bob may upload new one-time prekeys at anytime.

Bob will also upload a new signed prekey and prekey signature at some interval (e.g. once a week, or once a month). The new signed prekey and prekey signature will replace the previous values.

After uploading a new signed prekey, Bob may keep the private key corresponding to the previous signed prekey around for some period of time, to handle messages using it that have been delayed in transit. Eventually, Bob should delete this private key for forward secrecy (one-time prekey's private keys will be deleted as Bob receives messages using them; see Section 3.4).

3.3. Sending initial message.

To perform an X3DH key agreement with Bob, Alice contacts the server and fetches a “prekey bundle” containing the following values:

- Bob’s identity key IK_B
- Bob’s signed prekey SPK_B
- Bob’s prekey signature $Sig(IK_B, Encode(SPK_B))$
- (Optionally) Bob’s one-time prekey OPK_B

The server should provide one of Bob’s one-time prekeys if one exists, and then delete it. If all of Bob’s one-time prekeys on the server have been deleted, the bundle will not contain a one-time prekey.

Alice verifies the prekey signature and aborts the protocol if verification fails.

Alice then generates an ephemeral key pair with public key EK_A.

If the bundle does not contain a one-time prekey, she calculates:

$$DH_1 \equiv DH(IK_A, SPK_B)$$
$$DH_2 \equiv DH(EK_A, IK_B)$$
$$DH_3 \equiv DH(EK_A, SPK_B)$$
$$SK \equiv KDF(DH_1 || DH_2 || DH_3)$$

If the bundle does contain a one-time prekey, the calculation is modified to include an additional DH:

$$DH_4 \equiv DH(EK_A, OPK_B)$$
$$SK \equiv KDF(DH_1 || DH_2 || DH_3 || DH_4)$$

Note that DH_1 and DH_2 provide mutual authentication, while DH_3 and DH_4 provide forward
secrecy.

After calculating SK, Alice deletes her ephemeral private key and the DH outputs. Alice then calculates an “associated data” byte sequence AD that contains identity information for both parties:

$$AD \equiv Encode(IK_A) \| Encode(IK_B)$$

Alice may optionally append additional information to AD, such as Alice and Bob’s usernames, certificates, or other identifying information.

Alice then sends Bob an initial message containing:

- Alice’s identity key IK_A
- Alice’s ephemeral key EK_A
- Identifiers stating which of Bob’s prekeys Alice used
- An initial ciphertext encrypted with some AEAD encryption scheme [4] using AD as associated data and using an encryption key which is either SK or the output from some cryptographic PRF keyed by SK.

The initial ciphertext is typically the first message in some post-X3DH communication protocol. So it is the first message within some post-X3DH protocol, and also a part of Alice’s X3DH initial message.

After sending this, Alice may continue using SK or keys derived from SK within the post-X3DH protocol for communication with Bob, subject to the security considerations in Section 4.

3.4. Receiving the initial message

Upon receiving Alice’s initial message, Bob retrieves IK_A and EK_A from the message. Bob also loads the corresponding private keys of IK_B and SPK_B (and OPK_B, if Alice used it)

Now with these keys from Alice, Bob can repeats the exact same DH and KDF calculations from the previous section to derive the same SK, and then deletes the DH values.

Bob then constructs the AD byte sequence using IK_A and IK_B, as described in the previous section. Finally, Bob attempts to decrypt the initial ciphertext using SK and AD. If the initial ciphertext fails to decrypt, then Bob aborts the protocol and deletes SK.

If the initial ciphertext decrypts successfully the protocol is complete for Bob.

Bob deletes any one-time prekey private key that was used, for forward secrecy.

Bob may then continue using SK or keys derived from SK within the post-X3DH protocol for communication with Alice, subject to the security considerations in Section 4.

4. Security consideration.

4.1. Authentication

Before or after an X3DH key agreement, the parties may compare their identity public keys IK_A and IK_B through some authenticated channel.

Methods for doing this are outside the scope of this document.

If authentication is not performed, the parties receive no cryptographic guarantee as to who they are communicating with.

4.2. Protocol replay

If Alice's initial message doesn't use a one-time prekey, it may be replayed to Bob and cause Bob to think Alice had sent him the same message (or messages) repeatedly.

To mitigate this, a post-X3DH protocol may wish to quickly negotiate a new encryption key for Alice based on fresh random input from Bob. This is the typical behavior of Diffie-Hellman based ratcheting protocols [5].

Bob could attempt other mitigations, but analyzing these mitigations is beyond the scope of this document.

4.3. Replay and key reuse

Another consequence of the replays discussed in the previous section is that a successfully replayed initial message would cause Bob to derive the same SK in different protocol runs.

For this reason, any post-X3DH protocol MUST randomize the encryption key before Bob sends encrypted data. For example, Bob could use a DH-based ratcheting protocol to combine SK with a freshly generated DH output to get a randomized encryption key [5].

Failure to randomize Bob's encryption key may cause catastrophic key reuse.

4.4. Deniability

X3DH doesn't give either Alice or Bob a publishable cryptographic proof of the contents of their communication or the fact that they communicated.

Like in the OTR protocol [6], in some cases a third party that has compromised legitimate private keys from Alice or Bob could be provided a communication transcript that appears to be between Alice and Bob and that can only have been created by some other party that also has access to legitimate private keys from Alice or Bob (i.e. Alice or Bob themselves, or someone else who has compromised their private keys).

If either party is collaborating with a third party during protocol execution, they will be able to provide proof of their communication to such a third party. This limitation on "online" deniability appears to be intrinsic to the asynchronous setting [7].

4.5. Signatures

One might observe that mutual authentication and forward secrecy are achieved by the DH calculations. And so skip the prekey signature. However, this would allow a "weak forward secrecy" attack: A malicious server could provide Alice a prekey bundle with forged prekeys, and later compromise Bob's IK_B to calculate SK.

Alternatively, it might be tempting to replace the DH-based mutual authentication (i.e. DH_1 and DH_2) with signatures from the identity keys. However, this reduces deniability, increases the size of initial messages, and increases the damage done if ephemeral or prekey private keys are compromised, or if the signature scheme is broken.

4.6. Key compromise

Compromise of a party's private keys has a disastrous effect on security, though the use of ephemeral keys and prekeys provides some mitigation. Compromise of a party's identity private key allows impersonation of that party to others. Compromise of a party's prekey private keys may affect the security of older or newer SK values, depending on many considerations.

A full analysis of all possible compromise scenarios is outside the scope of this document.

4.7. Server trust

A malicious server could cause communication between Alice and Bob to fail (e.g. by refusing to deliver messages).

If Alice and Bob authenticate each other as in Section 4.1, then the only additional attack available to the server is to refuse to hand out one-time prekeys, causing forward secrecy for SK to depend on the signed prekey's lifetime.

This reduction in initial forward secrecy could also happen if one party maliciously drains another party's one-time prekeys, so the server should attempt to prevent this, e.g. with rate limits on fetching prekey bundles.

4.8. Identity binding

Authentication as in Section 4.1 does not necessarily prevent an "identity misbinding" or "unknown key share" attack. This results when an attacker ("Charlie") falsely presents Bob's identity key fingerprint to Alice as his (Charlie's) own, and then either forwards Alice's initial message to Bob, or falsely presents Bob's contact information as his own.

The effect of this is that Alice thinks she is sending an initial message to Charlie when she is actually sending it to Bob.

To make this more difficult the parties can include more identifying information into AD, or hash more identifying information into the fingerprint, such as usernames, phone numbers, real names, or other identifying information. Charlie would be forced to lie about these additional values, which might be difficult.

However, there is no way to reliably prevent Charlie from lying about additional values, and including more identity information into the protocol often brings trade-offs in terms of privacy, flexibility, and user interface. A detailed analysis of these trade-offs is beyond the scope of this document.

5. References.

[1] A. Langley, M. Hamburg, and S. Turner, "Elliptic Curves for Security." Internet Engineering Task Force; RFC 7748 (Informational); IETF, Jan-2016. http://www.ietf.org/rfc/rfc7748.txt

[2] T. Perrin, “The XEdDSA and VXEdDSA Signature Schemes,” 2016. https://whispersystems.org/docs/specifications/xeddsa/

[3] H. Krawczyk and P. Eronen, “HMAC-based Extract-and-Expand Key Derivation Function (HKDF).” Internet Engineering Task Force; RFC 5869 (Informational); IETF, May-2010. http://www.ietf.org/rfc/rfc5869.txt

[4] P. Rogaway, “Authenticated-encryption with Associated-data,” in Proceedings of the 9th ACM Conference on Computer and Communications Security, 2002. http://web.cs.ucdavis.edu/~rogaway/papers/ad.pdf

[5] T. Perrin, “The Double Ratchet Algorithm (work in progress),” 2016.

[6] N. Borisov, I. Goldberg, and E. Brewer, “Off-the-record Communication, or, Why Not to Use PGP,” in Proceedings of the 2004 acm workshop on privacy in the electronic society, 2004. http://doi.acm.org/10.1145/1029179.1029200

[7] N. Unger and I. Goldberg, “Deniable Key Exchanges for Secure Messaging,” in Proceedings of the 22Nd acm sigsac conference on computer and communications security, 2015. http://doi.acm.org/10.1145/2810103.2813616