vault1317/signal-dakez: An authenticated key exchange protocol with a public key concealing and a participation deniability designed for secure messaging

Richard B. Riddick

Vault Labs, Butcher Bay

root@hardenedvault.net

Abstract

A deniable authenticated key exchange can establish a secure communication channel while leaving no cryptographic evidence of communication. Some well-designed protocol today, even in the case of betrayal by some participants and disclosure of long-term key materials, cannot leave any cryptographic evidence. However, this is no longer enough: If "Big data" technology is used to analyse data fetched from pivotal nodes, it's not difficult to register your identity through your long-term public keys. (although it can't be a solid evidence due to deniability) In this article, we have analysed the advantages and disadvantages of existing solutions which are claimed to be deniable to some degree, and proposed an authenticated key exchange protocol that is able to conceal the public keys from the outside of the secure channel, and deniable to some degree, and a reference implementation is provided.

Keywords: key exchange, deniability, key concealing, secure messaging, OTR, Signal

Introduction of backgrounds

Over the past decade, electronic communication has become increasingly popular. The Internet has become a platform for the transmission of most critical discourse. At the same time, the revelation of wide-ranging surveillance and interference of the Internet in the field of security technologies has increased the public interest in communication security and privacy. Many people want to maintain control over their information online, but few are sure they can do it.

In response to this development, new secure messaging protocols, components, and applications have mushroomed. The purpose of these end-to-end encryption protocol is to establish encrypted session between endpoints of communications (and often to provide perfect forward secrecy on the basis of such session) so that messages inside such session are difficult to easily break from the outside. They also provide a mechanism for checking the integrity of the message and confirm that the message is not tampered with outside it. However, integrity mechanisms (e.g. signature) often provide undeniable evidence that someone do send a particular message. As a result, some of the teams who took the OTR developers as an example began to work to achieve the unification of integrity and deniability while communicating normally, which means that, at least provided that participants of the communication is able to confirm that the message is not tampered with, while leaving no cryptographic evidence of a participant's sending a certain message.

In order to enable both participant to authenticate each other's identities, these protocols usually use some form of digital signature. If the public key used to verify signature is sent to the other party in plain text, it causes the problems of **identity exposure**, i.e. This allows an adversary controlling the low-level protocol components on which end-to-end encryption protocols rely on communication (for example, the server relaying messages) to know **who is talking to whom**, even if they do not understand the content of the communication. In the background of large-scale censorship, the Big Brothers' target is not only the content, but also a broad interest in the metadata in order to draw a social graph. If the server getting compromised or the system administrator being evil are added into the threat model, the attacker can get a social graph of who is talking to whom without compromising the encrypted channel, and use it even more. In post-PRISM era, Big Brothers' "We kill target based on metadata." meant to be any one of us, or our potential malicous Turing machine.

OTR (v3 and v4) and Axolotl (signal protocol) are the most important end-to-end encryption protocols that ensure a certain deniability. Let's focus on these two protocols below.

OTRv3

The OTR development team is always committed to developing secure messaging protocol with deniability. They provide libraries implementing the protocol and tools for message forging to minimize the impact when intercepted ciphertexts being used as evidence. As far as the authors know, only the OTR series protocols have been able to claim that they have deniability, while specifically embracing a symmetric cipher mode of operation that is easy to forge and providing dedicated tools, but their protocols are carefully crafted to ensure the integrity of communication.

Currently, it is the third generation protocol which is mature and widely used. It has an important feature that is not noticeable, but important to implement its own deniability, and should be widely appreciated: the feature to conceal the public key.

OTRv3 must create its own session through the handshaking (authenticated key exchanging) process: the parties who communicate first establish an **unauthenticated** DHE key exchange (in which some technique is used to ensure that **none of the two is able to decide their own pre key according to the public part of pre key sent by their peer**). When the DHE is complete, with the shared secret available, the long-term public key (pub) , the initial value of the pre key counter (keyid), and a signature tied to both parties' pre key, pub and keyid, shall be **encrypted** with a key derived from the shared secret and then sent to the other party for verification to authenticate the **newly completed** DHE. With hiding the public key and the signature required for authentication the encrypted channel created through the shared secret, only the public part of the pre key exchanged during the handshaking is revealed to the surveillant controlling the lower-level protocol element, and the surveillant does not grasp the long-term public key of either party in communication. Furthermore, it is not possible to obtain signatures to prove that the communication partners have completed the handshaking process. When a man-in-the-middle attack is carried out to obtain these sensitive data in the handshaking process, it is almost impossible for the middle man to randomly generate the same pre keys as well as obtaining the long-term private key of either end. As a result, the man-in-the-middle attack of the handshaking process will almost certainly destroy the subsequent signature verification and make the handshaking process a failure, and both ends will be noticed.

Because the signature algorithm used by OTRv3 has no **participant deniability** (in more detail later), if the OTRv3 simply transmit the public key and the signature as plain text as in TLS, its deniability will in principle disappear.

During the communication phase after handshaking, OTRv3 avoid using self-authenticated AEAD algorithms (or modes of operation), but use independent message authentication code (MAC) to protect the integrity of a ciphertext message, and the MAC key indirectly results from a constantly revolving shared secret. In order to achieve deniability at this stage, any MAC key that has been used to verify a ciphertext, and whose source shared secret has been erased is sent out by attaching to a new message in plain form. The expected goal is not the opposite end, but the all-possible existence of the surveillants. Anyone who has received a MAC key can forge a message that passes verification, and libOTR, the main implementation of OTRv3 even provides dedicated tools for this purpose. In this way, the surveillant cannot prove that a certain message was sent by someone surveiled but was not forged by themself, thereby achieving deniability. Before the MAC key is published through a new message, only the sender and receiver of the message can calculate it from the shared secret, so that the recipient of the message can be sure that the message is indeed sent by their peer, thereby ensuring integrity.

The main problem with OTRv3 is the lack of participant deniability in the signature algorithm: during the handshaking phase, the signature is visible to both interacting parties. If one of the participants helps obtain identity information, including this signature (originally hidden in the encrypted channel during the handshaking), it becomes an evidence hard to deny.

In addition, proponents of the Axolotl protocol also think that OTRv3 are difficult to use, do not support "decrypting the same message many times", the handshaking process is too complicated and the result is

too expensive, and offline messaging is not supported. However, in fact, "decrypting the same message several times" is actually a huge security problem, and although the OTRv3 does not formally support offline messaging, until the program is closed, the message sent by the other party (encrypted after performing a DHE with the public part of pre key inside the last sent message) can be decrypted as usual in the event of a temporary drop-off.

Axolotl a.k.a "Signal Protocol"

Axolotl is a comprehensive end-to-end-encryption protocol, invented by Open Whisper Systems, first applied for Signal, then adjusted to the XMPP in the form of omemo and absorbed by the Matrix Protocol (modified as OLM), including the digital signature algorithm XEdDSA, the handshake protocol X3DH , message encryption protocol Double Ratchet, session management protocol Sesame, and other sub-protocols. The protocol was later renamed "Signal Protocol", but because "Signal Protocol" is not "Signal's Protocol (all communication protocols that make up the Signal)", we are going to use Axolotl in this article to refer to this protocol.

From the point of view of application, the main of features which Axolotl has more than OTR is offline messaging - that includes two aspects: on the one hand, completing a handshake does not require both parties to be online at the same time, and on the other hand, the communication process that follows the handshake does not require both communication parties are online at the same time. Proponents of the Axolotl Protocol argue that this significantly increases the usability of the protocol, but we still believe that the way the Axolotl Protocol implements offline messages heavily undermines confidentiality and deniability.

The OTRv3 described above requires a 4-way handshake to conceal the public key, so both parties must be online to complete the handshake in a sufficiently short time, otherwise it is better that the handshake starts from the beginning instead of waiting for a handshake message. To implement offline messaging, Axolotl does not only compress the handshake into 2-way, but also publishes the first handshake in the form of "Pre key bundle" on the server and connects the second handshake to the first message. Sent but not received messages are cached on the server and delivered when the receiving end goes online. There is a pre key signed with the public key and several unsigned pre key in the bundle, each pre key has its own number ("id") and each unsigned pre key can only be used once, but the entire pre key bundle allows several participants to handshake independently with the same publisher of the bundle.

X3DH is unable to apply any protection to public keys

But just because of this, X3DH is unable to apply any protection to public keys: The party issuing pre key bundle does not know how the other party will generate pre key and thus cannot encrypt their own public key for the other party; the sender of the first message cannot encrypt the public key, for they should allow the peer directly use their own public key and pre key to perform DHE. In this way, the server operator can easily collect correspondences between public key and user identity, and the signed pre keys are indisputable proof that the two parties are communicating (The Appendix A of dakez-popets18.pdf also sets out the method to attack Axolotl Protocol that also lacks participant deniability). Since the pre key bundle must be published to the server, protocols in which the server does not offer the functionality (e.g. IRC) can not adapt to Axolotl, although this problem may be solved through turning the offline handshake to an online one, that is, sending a simple bundle containing only one signed and one unsigned pre key as a handshaking message directly to the other party, but this adaption of the Axolotl, which can only be done online, has no advantage over OTRv3's public-key-concealing online handshake.

Double Ratchet session contexts stored in secondary storage are prone to stealing and forensic

Compared to OTRv3, which requires a DHE every time a message is sent, Axolotl only generate a new DH pre key and trigger a new round DHE once the other party sends the previous message. If one party continuously sends messages, only the key-derivation function (KDF) is used to update the session key. Indeed, this

approach can reduce the amount of calculation and improve the synchronization of session context at both ends. However, to satisfy the requirement for offline messages, Axolotl saves the context to the secondary storage (e.g. disk) every time it changes. The original intention was that even if the client goes offline or even crashes unexpectedly when the client goes back online, encrypted text messages cached on the server can still be decrypted. However, it also means that if context stored in the secondary storage is stolen, the attacker can decrypt messages to be received by the client instance continuously until it is interrupted by DH pre key generation triggered by message sending, and offline session context can also be a kind of proof that is difficult to deny.

OTRv3, on the other hand, does not store any session context components into the secondary storage. Until the session is finished and the client goes offline, the session context will be naturally destroyed; temporary network-induced disconnect can be automatically restored.

This problem can be solved by further refining the session lifecycle and strictly limiting archived data. The principle is to save data you have to save, such as the IDs and the secret part of bundled pre keys - because you cannot establish sessions without saving these data - and once you have successfully established a session, you will keep the session context only in RAM until the client goes offline, terminating the session. Later, if you need to "continuously" send offline message, you can complete it by creating a new session. (This policy means that converting an online session to an offline one is prohibited, while converting an offline session to an online one is allowed). An even stricter strategy could be used: strictly distinguish between online sessions and offline ones. The session created when the other party is online is an online session and it lasts until either party goes offline. As long as the client continues to run, the network-induced outage can be automatically restored. If the other party is not online when you create the session, the session is an offline one that lasts as long as one's own party goes offline or the other party goes online. For the offline side, they only need to receive (as described above, handshaking is completed when the first message is received and an offline session is created) the cached messages after you go online and then the session could be finalized. If communication need to be continued, a new online session could then be created (This strategy is equivalent to disabling the mutual conversion of online and offline sessions). This strategy can minimise sensitive data leakage via secondary storage while maintaining the ability of offline messaging.

The immutable client id intensifies identity exposure

To distinguish between different instances of the same account, Axolotl randomly creates a specific number for each instance (called "device id" in libsignal-protocol-c). Clients use this id to distinguish ciphertext message sent from different instance of the same account and send them to the appropriate session context for decryption. Therefore, the instance id itself must appear as plaintext so that the recipient can determine from which instance the ciphertext originates before decryption. In addition, Axolotl provides that account instances who do not change their id but only change their public key cannot be trusted, which means that the combination of instance id and public key can almost completely mark the instance of the account. Since public keys cannot be concealed in the two-way handshaking of X3DH, this combination must also appear as plaintext in the pre key bundle and the first message, which further facilitates the collection and account instance identification of the adversary controlling of low-level protocol components.

By contrast, the OTRv3 uses randomly generated "Instance tags" during the handshaking to mark different instances. Obviously, the instance tag can only be used to distinguish between different instances temporarily, but it cannot permanently mark an instance.

From an application point of view, instance id is useful to permanently identify a client instance, but it should be hidden in an encrypted channel and a method similar to that of an instance tag of OTRv3 is used to distinguish between different instances and participants before decrypting a message. To avoid breaking deniability, the compliance of instance tag and instance id should be maintained locally by the client.

The draft of OTRv4

Shortly after the publication of the above dakez-popets18.pdf, the OTR project team began developing a fourth generation protocol with the aim of further strengthening the deniability of the protocol, especially

in order to address the problem of the lack of **participant deniability** of OTRv3. The key to the online deniability of OTRv4 is the application of the **Ring Signature Algorithm** (in more detail later): In short, the ring signature algorithm uses **a set of public keys** and **one** of the corresponding private key to sign a payload, and the same set of public keys is used to verify. The result is that the private key that signed the payload can prove to be the private key corresponding to one of these public keys, but it is not possible to determine the exact key which made the signature. By carefully choosing public keys to be used in signing, both communication parties are able to determine each other's identity (because private keys corresponding to public keys other than the other party's identity key are held by theirself, but they are not used to generate signature, the signature can only be generated by the other party), but a participant cannot prove to others that the other party participates the conversation (because their own public keys are participant of the signature too, which renders the participant unable to prove that the signature is not signed with their own private key).

OTRv4 supports both online (3-way) and offline (2-way, similar to Axolotl) handshaking. Among them, the online handshaking is deniable for both communication parties, while the offline handshaking is not deniable for the party sending the first message to complete the handshaking. Ring signature is used in both handshaking modes.

During the communication phase, OTRv4 uses Double Ratchet to iterate its session context, but never saves the context to the secondary storage. OTRv4 also supports offline session initiated with pre key bundles, but only saves data necessary to completing handshaking with the first message to the secondary storage (This equals to forbidding to convert online sessions to offine ones, but OTRv4 allows to convert offine sessions to online ones).

The main problem with OTRv4: Maybe because the ring signature algorithm has participant deniability, OTRv4 does not implement public key concealing. This does reduce the number of handshaking ways, but also exposes public keys out of the encrypted channel, easing the adversary controlling the low-level protocol components collecting data and recognizing account instances. Although it does not affect deniability so much, it still cannot help prevent the adversary controlling the low-level protocol components from collecting metadata including public keys in order to draw social graphs.

A survey for various e2e encryption protocols

Protocol	Major Implementor	PFS	Algorithm on which DHE is based	Algorithm for User Authentication
OpenPGP	GnuPG	No	No	various signature (rsa, dsa, ecdsa, eddsa)
OTRv3	libotr	Yes	DHE on an 1536-bit MODP group in RFC 3526	DSA
axolotl/signal protocol	libsignal-protocol-c	Yes	X3DH based on x25519	XEdDSA based on Ed25519
olm	libolm	Yes	Customed X3DH	EdDSA with independent keys
omemo	libomemo	Yes	axolotl X3DH	axolotl XEd25519
Proteus	Wire	Yes	Curve25519	axolotl XEd25519
OTRv4	libotr-ng	Yes	DHE on Ed448 as well as the 1536-bit MODP group in OTRv3	EdDSA and ring signature on Ed448
vault1317	WIP	Yes	dakez based on Curve25519	XEdDSA and ring signature on Ed25519

Protocol	Symmetric Ratchets	Support offline messaging	Anonymity perserving (identity keys will not leak outside the participants)	symmetric algorithm
OpenPGP	N/A	Yes	No	CFB mode with various Block cipher
OTRv3	N/A	No	Yes	AES-CTR with detachable hmac-sha256
axolotl/signal protocol	axolotl Based on HKDF	Yes	No	AES-CBC
olm	olm and magolm	Yes	No	AES-CBC
omemo	axolotl for session key	Yes	No	AEC-CBC for axolotl, AES-GCM for message body
Proteus	Similar to axolotl	Yes	No	chacha20
OTRv4	Based on HKDF	Yes	No	chacha20
vault1317	same as axolotl	Yes	Yes	same as axolotl

Protocol	Deniability	Allow simultanous multiple recipients	Have Interactive (synchronous) AKE	Have pre key-based offline (asynchronous) AKE
OpenPGP	Weak	Yes	No	No
OTRv3	Online Very Strong (non-aead cipher, MAC revealing and tools provided)	No	Yes	No
axolotl/signal protocol	Weak (CBC cipher, no MAC revealing, no tool)	Yes	No	Yes
olm	Similar to axolotl's	magolm	No	Yes
omemo	Weak (aead cipher)	multiple encrypted session keys in one xml stanza	No	Yes
Proteus	non-aead cipher but no tool	encrypt the plain text message for every recipient	No	Yes

Protocol	Deniability	Allow simultanous multiple recipients	Have Interactive (synchronous) AKE	Have pre key-based offline (asynchronous) AKE
OTRv4	Very Strong (non-aead cipher, MAC revealing and tools provided)	WIP	Yes	Yes
vault1317	WIP	omemo	Yes	Yes

vault1317/signal-dakez

We believe that the public key concealing nature of OTRv3 is essential to protect communicator's privacy, so we decide to implement a protocol which is capable to conceal public keys, authenticate identities with ring signature, thus has participant deniability to some degree, has online and offline handshaking in the same time, and maintains session context with the way of Double Ratchet.

Improvement

In view of the complexity of the Double Ratchet, we are currently using the existing Double Ratchet implementation of the Axolotl and use existing components of the Axolotl protocol as far as possible for the implementation of the online and offline handshaking protocol with the characteristics necessary for us. In general, in addition to the original X3DH, we will write an online and an offline handshaking protocol based on one implementation (libsignal-protocol-c) of the Axolotl protocol, and use the key materials enchanged via these handshaking protocols to initialize the Double Ratchet subprotocol.

Here we will introduce the used protocol components and some of their implementation detail.

Digital Signature Algorithm xed25519

xed25519 is an algorithm modified from DJB's ed25519, using public and private keys for x25519 to produce digital signature, initiallly invented as a part of Axolotl, to be used in our protocol.

Implementing ring signature algorithm (in more detail later) also needs components of this algorithm - addition and subtraction of points on the elliptic curve Edwards25519.

Terminology

CurveM (e.g. Curve25519, Curve448) is a Montgomery curve defined on F[pˆ2], where M is a description of some characteristics of prime number p, for example, the p of Curve25519 is 2ˆ255 - 19; *xM* is the ECDH scheme implemented on CurveM, based on "absolute value of the product of a point and a scalar"; *edM* is the EdDSA algorithm implemented on the corresponding (twisted) Edwards curve (discretized on F[p] in homogeneous coordinates form, with rational points as its group elements); *EdwardsM* is the (twisted) Edwards curve itself used to implement edM; *Scalar*: The points on elliptic curves defined on discrete fields with their addition operation form a cyclic group. Assuming its order is prime number q (nearly all DH-related cryptographic scheme requires using cyclic groups with such property), a non-negative integer less than q is defined as a scalar. Via the operation to multiply a scalar with the base point (a generator of the cyclic group) G, scalar a and point A on the elliptic curves form an one-to-one correspondence: A = a * G.

curve25519 and X25519

curve25519 and edwards25519 is two different forms of elliptic curve with mathematical link, among which, curve25519 is a Montgomery curve (all Montgomery curve can be transformed into (twisted) Edwards curve, and elliptic curves in "Montgomery form" can all be transformed into standard Weierstrass form, so "Montgomery" should be seen as a property rather than a representation) v^2 = (u^3 + 486662u^2 + u) mod p, p = (2^255 - 19)^2, as a cyclic group, the order is 8q, q = 2^252 + 27742317777372353535851937790883648493, so it is isomorphic with the multiplication group modulo (8q + 1), and (8q + 1) should also be a prime; its subgroup with order q is used in cryptography.

The origin of its parameter and other properties could be seen in RFC7748.

The equation of curve25519 shows that the curve is symmetrical to the horizontal axis (u). Combined with the property of the additive group of the points, if Q=(u,v) is a point on the curve, O is the point at infinity (the identity element of the additive group of the points on elliptic curve, a.k.a, the zero element), then the coordinate of -Q = O-Q is (u,-v). Therefore, only the value of the abscissa and the sign of ordinate must be saved to unambiguously express the point on elliptic curve in Montgomery-Weierstrass form unequivocally, including curve25519.

What's more, the abscissa of a point has a characteristic similar to absolute value, so we could define abs(Q) = abs(-Q) = u .

Curve25519, as an ECDH implementation (X25519), actually implements a fast algorithm with known "absolute value" of a point abs(A), and scalar m to compute abs(mA); there is no unambiguous expression of points themselve in the algorithm of X25519, only expression of the "absolute value" namely abscissa:

The X25519 scheme maps a private pre key/scalar, to another "abs" of a point, to another "abs" of a point, abs(mG), which is the public part of the pre key, through an operation to compute the absolute value of the multiplication of a point and a scalar, with the true generator G=(9,+) (in reality, only the absolute value abs(G)=9 is saved). (in reality, -G=(9,-) is also a generator, but because abs(mG)==abs(m(-G), it doesn't matter in application.)

Addition and subtraction operation are unable to implement with such representation: Only the knowledge of abs(A) and abs(B) cannot construct an algorithm to calculate abs(A+B) or abs(A-B). Although X25519 provides an algorithm to compute abs(2A) and abs(A+B) from abs(A), abs(B) and abs(A-B) (This algorithm is mainly used to compute products in a constant-time manner via Montgomery ladder algorithm), but abs(A-B) cannot be obtained from only abs(A) and abs(B).

Since the construction of DSA-type algorithm on elliptic curves requires the addition and subtraction of points, the functional elements of X25519 alone cannot be used to construct such an algorithm. Additionally, the Small subgroup attacks and invalid curve attacks) has been considered by the modern implementation of curve25519 but due to the above-mentioned problem of the space reduction of abs(aG), the current best practice still follows:

- x25519 key pair for DH key agreement, while ed25519 key pair for signature only

If you need to use a single ed25519 key pair for both DH and signature, you can convert them to x25519 form on demand, eg: root key is ed1, y2u() can convert both priv/pubkey to x25519 form, which can be used for key exchange and encryption to meet e.g: TLS use case . We must consider two factors: 1) the acceptable degree of performance loss 2) the security of the conversion function implementation, currently you can refer to the relevant implementation of libsodium, where the scalar s needs to be extracted during the process of converting the ed25519 private key to the x25519 private key.

Edwards25519

Ed25519 refers to a deterministic DSA-type algorithm implemented by DJB on the twisted Edwards curve corresponding to curve25519. In this article, Edwards25519 is used to refer to this twisted Edwards curve itself:

y^2 - x^2 = 1 - (121665/121666)x^2 *y^2

Different with Montgomery form, the coordinates of points on (twisted) Edwards curve are no longer integers but rational numbers, and the discretization in this form can only be done under homogeneous coordinates: setting x = X/Z, y = Y/Z, xy = T/Z, X, Y, Z, T are integers, then

121666Z^2 (Y^2 - X^2) = 121666Z^4 - 121665X^2 *Y^2 mod p is the corresponding discrete form, and {X:Y:Z:T} is used to represent a point on the curve.

As a cyclic group, the order is the same 8q, and the subgroup with order q is also used to implement cryptography.

Note: During discretization, the Montgomery form is defined on F[P^2], while the (twisted) Edwards form is defined on F[p].

The mathematical link between edwards25519 and curve25519 guarantees that most points on curve25519 could be mapped to points on edwards25519, represent as mont2ed(u,v); there is a function u2y(u) = (u - 1)/(u + 1) able to easily convert the abscissa of a point on curve25519 to the ordinate of the mapped point on edwards2551, and mont2ed(u,v)=(u2y(u)/v,u2y(u)).

Similar to curve25519, the symmetry of edwards25519 makes the representation of a point on the curve only requires to save its ordinate and the sign of its abscissa. That is to say, if R=(+,y) then -R=(-,y).

Because edwards25519 provides an unambiguous representation of a point on the curve, the addition and subtraction of points and then digital signature similar to DSA can be implemented. The true multiplication between a scalar and a point (contrary to "absolute value of product" operation on curve25519) is of course available, thus also capable to be used for ECDH.

The generator B=(+,4/5) on edwards25519 is exactly the result of the generator G=(9,+) of curve25519 mapping into edwards25519: B = mont2ed(G) = (+,u2y(abs(G)).

ed25519-modified

In DJB's EdDSA paper, the relationship between the public and private key is not linear: "Standard" EdDSA uses an arbitrary byte string with certain length as a private key, which is hashed to obtain 64 bytes of binary data, of which the low 32 bytes are converted to a scalar capable to operate on edwards25519: s = hash_to_scalar(sk), and the public key A = sB; while the high 32 bytes are used as a seed with the signed data m to generated the "random" scalar k used in DSA type algorithm.

Signal modifies it into ed25519-modified, in which the scalar s is directly used as the private key, and also used as a seed to generate k. Its advantage is the linear relationship between the public and private key is kept, while its drawback is that it increases the correlation between k and scalar s (This problem may be solved by generating k from s and m using rfc6979. Currently Signal improves it by introducing another random seed **rand** externally, making k unambiguously determined by (s, m, rand)).

Convert public and private keys between curve25519 and edwards25519 forms

We define sgn(R) as a function to get the sign of (the abscissa of) a point on curve edwards25519.

u2y() alone is not enough to determine the "plus or minus" of a mapped point, so XEdDSA uses below scheme to map the public and private part of pre keys on curve25519 to the public and private keys on ed25519-modified:

conv_mont_pubkey(abs(A)) = (+,u2y(abs(A)));

It is mandatory that the "positive" mapping result of the public part of pre key is selected as the public key of ed25519-modified.

conv_mont_privkey(a) = { if(sgn(aB)==(+)) return a; else return ((-a) mod q); }

(q is the order of the additive group described above on curve25519 and edwards25519)

If the multiplicative product of a private pre key a of curve25519 and the generator B of edwards25519 is actually "negative", its additive inverse ((-a) mod q) = q - (a mod q) is chosen as the private key of ed25519-modified.

This mapping method is self-assured, because it guaranteed B=conv_mont_pubkey(abs(G)) and conv_mont_privkey(a)B=conv_mont_pubkey(abs(aG)), but this method reduces the capable space of private and public keys (from the price of X25519 discarding the sign of points).

This mapping rule is hidden in the logic of XEd25519, though not directly used explicitly, but the ring signature algorithm based on it will directly use this rule.

The consideration of XEd25519

XEd25519 makes use of the unoccupied space to save the sign of point **aB**, which is the result of mapping the scalar **a** into edwards25519 curve:

XEd25519-sign(m, a, (rand)) = (ed25519-modified-sign(m, a, (rand)), sgn(aB)) ,

During verification, if aB's sign saved in signature is negative, indicating conv_mont_pubkey(abs(A)) == -aB, then the additive inverse of conv_mont_pubkey(abs(A)) will be used to verify the signature:

XEd25519-verify(abs(A=aG), (sig, sgn)) = { if(sgn == (+)) return ed25519-modified-verify(conv_mont_pubkey(abs(A)), sig); else return ed25519-modified-verify(-conv_mont_pubkey(abs(A)), sig); }

Conditional logics should be replaced with constant-time logics during implementation, in order to prevent side-channel attacks and differential analyses.

rsig-(x)ed25519

For this implementation, the curve25519 and ed25519-modified codes integrated with libsignal-protocol-c are used. The ring signature algorithm rsig/rvrf discussed in DAKEZ. These types of algorithms use **a set of public keys** and the payload signed by one of the corresponding private key, while signature verification uses the same set of public keys. It is designed to prove that the private key used for the signature is one of the private keys corresponding to the set of public keys, but it is impossible to determine the private key with which the signature is signed.

The ring signature algorithm can be implemented on the basis of RSA or DSA (generally, including DSA on a elliptic curve, such as ECDSA and EdDSA). The DSA-based algorithm has a concise mathematical structure, and the same parameters is easy to integrate the same type of algorithm with DH and single-key DSA algorithms.

The common elements of DSA species

The ring signature algorithm being used in this project is highly related to DSA, so it must use many of the mathematical elements required by the DSA-type algorithm.

Scalar fields Fields composed of non-negative integers less than a prime number q. All four arithmetic operations are accompanied by an operation to modulo q, and its elements are called scalars (indicated by lowercase letters below). The private key is a randomly selected non-zero scalar.

Cyclic group In the cyclic group described in the above quotation, its elements form a one-to-one mapping with the elements in the scalar fields through the generator G and the operation '^'. This mapping relationship is generally expressed in the form of a power, but on an elliptic curve it is implemented as a multiplication of a point and a scalar, that is, the point and itself are repeatedly added, and the number of repetitions is the value of the scalar.

Operation fields Elements and operations in a cyclic group are often implemented on a fields that consists of non-negative integers less than the prime p (p > q) . Note: The operation fields and the scalar fields are not the same. Operations on the operation field cannot be used to compute elements of the scalar field.

The libsignal-protocol-c's implementation of mathematical objects in ed25519-modified

In this implementation, the external representation of the scalar field and the operation field (that is, the representation during storage and transmission, is opposite to the internal representation, which is the representation during operation. For example, "network byte order" is an external representation; "host byte order" is an internal representation) are 256-bit integers represented by a 32-element array of 8-bit unsigned integers (value(a) = i from 0 to 31 sum a[i]*256^i).

The internal representation of an element of the scalar field is the same as its external representation, and the prime modulo of the four arithmetic operations is q = 2^252 + 27742317777372353535851937790883648493.

The internal representation of an element of the operation field is a large integer represented by a 10-element array of 32-bit signed integers (see fe.h for its definition). The modulo of the four arithmetic operations is p = 2^255-19, and the edwards25519 curve is defined on the finite field F[p] with modulo p (the corresponding Montgomery form curve25519 is defined on F[p^2]), as a cyclic group its real order is 8q (so it is isomorphic to the integer multiplicative group modulo 8q+1. As a comparison, the order of the integer multiplicative group modulo 7=6+1 is 6, which contains subgroups of order 2 and order 3, because 6=2*3). Cryptography uses the order q subgroup, whose generator is (+, 4/5).

Because the calculation of points on the elliptic curve is generally carried out under projective coordinates, the elements of the cyclic group, that is, the points on the elliptic curve edwards25519, are generally represented internally by projective coordinates (such as (X:Y:Z) satisfying x=X/Z, y=Y/Z represents the coordinates of the point (x, y), where X, Y, and Z are all elements of the operation field using internal representation). Obviously, the internal representation of the same point on the curve is not unique (The internal representation of a point has more than one form, see ge.h).

The external representation can represent up to 2^256-1, and only a value greater than 2^255 will make the highest bit (that is, the highest bit of a[31]) 1, and as described in the xed25519 chapter, it unambiguously represents the point on edwards25519 only need to save the value of the ordinate and the sign of the abscissa, because the ordinate is an element of the operation field, and the largest element in the operation field is p-1 = 2^255-20 <2^255, that is, the highest bit of the external representation of all elements of the scalar field and operation field is 0, so the highest bit of the external representation can be used to store the sign of the abscissa. In this way, the three different mathematical objects are unified into the same external representation.

Ring signature

RSig(ai, {A0, A1, A2}, m), i = 0 or 1 or 2

- Calculate the public key Ai = G^ai corresponding to the private key and judge the value of i. If i does not belong to {0, 1, 2}, it will fail;
- Without loss of generality, assuming i = 0, the following values `t0, c1, c2, r1, r2` are randomly selected in the scalar field;
- Calculate T0 = G^t0, T1 = G^r1 * A1^c1, T2 = G^r2 * A2^c2; (Note: here * operation is the operation of two elements in the cyclic group, which is realized as the addition of points on the elliptic curve)
- Calculate c = ToScalar(Hash(usageAuth || A0 || A1 || A2 || T0 || T1 || T2 || m)), that is, intercept a part of the hash value as an integer and take the modulus to obtain an element c in the scalar field;
- Calculate c0 = c-c1-c2 (mod q);
- Calculate r0 = t0-c0 * a0 (mod q);
- sig = ((c0, r0), (c1, r1), (c2, r2)) is the ring signature value;

The signature verification function Rvrf({A0, A1, A2}, m, sig = ((c0, r0), (c1, r1), (c2, r2))) is symmetric:

- Calculate Ti = G^ri * Ai^ci;
- Calculate c = ToScalar(Hash(usageAuth || A0 || A1 || A2 || T0 || T1 || T2 || m));
- Judgment whether c == c0 + c1 + c2 (mod q)

prove: Because r0 = t0-c0 * a0 (mod q) i.e. t0 = r0 + c0 * a0 (mod q) then T0 = G^t0 (mod q) = G^(r0 + c0 * a0) (mod q) = G^r0 * G^(c0 * a0) (mod q) = G^r0 * (G^a0)^c0 (mod q) = G^r0 * A0^c0 (mod q)

This algorithm can easily be extended to the case of i = 1 or 2, see rsig.c for details, where conditional logic is replaced by constant-time logic.

Interface with libsignal-protocol-c

The implementation in rsig.c is based on edwards25519, but all the asymmetric keys in libsignal-protocol-c uses the curve25519 form as its external representation, so the mapping methods conv_mont_pubkey() and conv_mont_privkey() mentioned in the xed25519 chapter are needed to convert the form of curve25519 to edwards25519 which can be used for calculation.

vault1317/signal-dakez

definition:

Symbols	Description
`G`	Generator of the cyclic group
`G^k`	The public part of a DH (pre) key corresponding to the private part of the (pre) key k
`G^xy`	Result of DH key exchange starting from private parts of pre keys x and y
`Rng()`	The output of the random number generator
`SymE/SymD(k, m)`	Use k to encrypt/decrypt data m with no more than one block (if it is a block cipher) with a symmetric algorithm Sym
`SymSE/SymSD(k, m)`	Use key to encrypt/decrypt data m of arbitrary length with a certain working mode of symmetric algorithm (if it is a block cipher) SymS
`a||b`	Data block concatenation operation
`{a,b,...}`	Parsable data serialization
`KDF(p, d)`	Use the algorithm KDF to derive the key from the parameter p and the data d
`I[Alice]`	DH (private) key used to mark Alice's identity
`ID[Alice]`	Alice's registration ID
`FD[alice]`	Alice's temporary device ID
`E[Alice]`	The (private part of) DH pre key generated by alice
`Dgst(m)`	digest of data m
`Mac(k, m)`	Message authentication code of data m signed with authentication key k

Symbols	Description
`FP(G^k)`	The fingerprint of the DH public key of the private key k, generally implemented by the message digest algorithm
`DSig(k, m)`	DSA-type algorithm for signing data m with private key k
`DVrf(G^k, m, dsig)`	verification function corresponding to DSig
`Rsig(x, G^x, G^y, G^z, m)`	Ring signature algorithm for signing data m with private key x and public key G^x, G^y, G^z
`RVrf(G^x, G^y, G^z, m, rsig)`	RSig corresponding verification function
`(r[0], c[0]) = SCDK(ss)`	The initial root key and chain key are generated from the shared secret
`(r[n+1], c[n+1][0]) = RKCC(r[n], s)`	Iterate the root key and generate a chain key
`c[n+1] = CKCN(c[n])`	chain key self iteration
`m[n] = CKMK(c[n])`	Export message key from chain key

The ring signature algorithm must meet: If w, x, y, x are not equal, then

RVrf(G^x, G^y, G^z, m, Rsig(x, G^x, G^y, G^x, m)) == RVrf(G^x, G^y, G^z, m, Rsig(y, G^x, G^y, G^x, m)) == RVrf(G^x, G^y, G^z, m, Rsig(z, G^x, G^y, G^x, m)) == true

then

RVrf(G^x, G^y, G^x, m, Rsig(w, G^x, G^y, G^z, m)) == false

signal-IDAKE (Online interactive deniable key exchange)

If both communication parties are online, signal-IDAKE can be used to initialize the basic material required for Double Ratchet through 5-way handshakes. Noted that all data except public parts of pre keys G^E[a] and G^E[b] are hidden in encrypted channels, and a ring signature algorithm is applied to authenticate the identity.

Before the handshake, Alice and Bob hold their private part of identity keys I[a] and I[b] respectively, as well as the constants defined in the protocol, such as G, C [0] ~ C [7]. Alice initiates signal-IDAKE with Bob as follows:

- Alice
  1. Generate E[a]=Rng(), calculate G^E[a] and D[a]=Dgst(G^E[a]);
  2. Send D[a] to Bob;
- Bob
  1. Generate E[b]=Rng(), calculate G^E[b];
  2. Send G^E[b] to Alice;
- Alice
  1. Calculate S=G^E[a]E[b], k[0]=KDF(C[0], S), k[1]=KDF(C[1], S), k[2] =KDF(C[2], S);
  2. Send {G^E[a], CE[0]=SymE(k[0], {G^I[a], ID[a]})} to Bob;

Neither party can generate its own pre key based on the other party's pre key as vault1317/signal-dakez follow the OTRv3 approach.

- Bob
  1. Verify Dgst(G^E[a])==D[a];
  2. Calculate S=G^E[a]E[b], k[0]=KDF(C[0], S), k[1]=KDF(C[1], S), k[2]=KDF(C[2], S);

3. Calculate {GˆI[a], ID[a]}=SymD(k[0], CE[0]);
4. Calculate T=KDF(C[3], GˆI[a] ||GˆI[b]);
5. Calculate P[b]=Rsig(I[b], GˆI[a], GˆI[b], GˆE[a], (C[4]||ID[a]||ID[b]||GˆE[a] ||GˆE[b] ||T));
6. Send to Alice CE[1]=SymSE(K[1], {GˆI[b], ID[b], P[b]});

- Alice
  1. Calculate {GˆI[b], ID[b], P[b]}=SymSD(K[1], CE[1]);
  2. Verify RVrf(GˆI[a], GˆI[b], GˆE[a], (C[4]||ID[a]||ID[b]||GˆE[a] ||GˆE[b] ||T), P[b])==true;
  3. Calculate T=KDF(C[3], GˆI[a] ||GˆI[b]);
  4. Calculate P[a]=Rsig(I[a], GˆI[a], GˆI[b], GˆE[b], (C[5]||ID[a]||ID[b]||GˆE[a] ||GˆE[b] ||T));
  5. Send to Bob CE[2]=SymSE(k[2], P[a]);

The ring signature here uses three public keys. For Alice, because the private keys I[a] and E[a] are both in her possession, she can be sure that P[b] is exactly issued by Bob using I[b], but because of this, she cannot prove to others that P[b] was not signed by herself with I[a] or E[a]. As a result, communication participants can use this to determine the identity of the other party, but they cannot prove it to a third party. The authentication information exchanged in the last three handshake is respectively encrypted with three symmetric keys derived from the shared secret and then transmitted to prevent them from leaking out of the session.

- Bob
  1. Calculate P[a]=SymSD(k[2], CE[2]),
  2. Verify RVrf(GˆI[a], GˆI[b], GˆE[b], (C[5]||ID[a]||ID[b]||GˆE[a] ||GˆE[b] ||T), P[a])==true;

So far, Alice and Bob have received the same shared secret S, the peer's pre key GˆE[b] and GˆE[a] and the public part of identity key GˆI[b] and GˆI[a]. The shared secret and pre key can then be used in the Double Ratchet protocol.

signal-ODAKE (Offline interactive deniable key exchange)

When one of the two communicating parties is offline, and the online party holds the other party's long-lived authentication public key beforehand, this protocol can be used to initiate an offline session to leave a message to the other party. The offline message is cached on the server and received after the receiver goes online. In order to establish an offline session, it is necessary to publish some pre keys on the server, but for the sake of privacy, the pre key bundle **does not include** the public part of the long-term identity key of the publisher.

Before the handshake, Alice already holds their own private part of the identity key I[a], and Bob's public part of the identity key GˆI[b] (such as the cached result after a signal-IDAKE is completed). Alice will initialize signal-ODAKE with Bob as follows:

- Bob
  1. Generate Q[b,j]=Rng(), and optional one-time pre keys E[b,0]=Rng(),... and calculate D=DSig(I[b], GˆQ[b,j])
  2. Publish the pre key bundle {ID[b], j, GˆQ[b,j], D, GˆE[b,0],... } to the server. Note: In stage 2, ID[b] is replaced with FD[b], in order to guarantee that ID never appears as plain text. The rest are unchanged.
- Alice
  1. Generate E[a,0]=Rng(), E[a,1]=Rng(),
  2. Because Bob is offline, signal-IDAKE cannot be used, so get the pre key bundle published by Bob from the server, and select one of the one-time pre keys GˆE[b,i], and its subscripts i, and GˆQ[b,j] corresponds to the subscript j,
  3. Verify that DVrf(GˆI[b], GˆQ[b,j], D)==true,
  4. Calculate S=(GˆE[b,i]E[a,0] ||GˆQ[b,j]E[a,0] ||GˆI[b]E[a,0]), k[0]=KDF(C[6], S), k[1]=KDF(C[7], k[0]) So far, Alice has obtained the shared secret S and a pre key GˆE[b,i]
  5. Calculate T=KDF(C[3], GˆI[a] ||GˆI[b]), M=Mac(k[1], (C[5]||i||ID[a]||ID[b]||GˆE[a,0] ||GˆE[b,i] ||T)), P=Rsig(I[a], GˆI[a], GˆI[b], GˆE[b,i], (C[5]||i||ID[a]||ID[b]||GˆE[a,0] ||GˆE[b,i] ||T)), CE=SymSE(k[0], {GˆI[a], ID[a], M, P}),
  6. Assemble H={GˆE[a,0], i, j, CE}

7. Send {H, payload. . . }, where payload is the intact encrypted message generated by the Double Ratchet protocol initialized with S and E[a,1], GˆQ[b,j].

- Bob (Getting online)
  1. Find E[b,i] and Q[b,j] from the local session storage according to i, j
  2. Calculate S=(GˆE[a,0]E[b,i] ||GˆE[a,0]Q[b,j] ||GˆE[a,0]I[b]), k[0]=KDF(C[6], S), k[1]=KDF(C[7], k[0])
  3. Calculate {GˆI[a], ID[a], M, P}=SymSD(k[0], CE), T=KDF(C[3], GˆI[a] ||GˆI[b]),
  4. Verify that M==Mac(k[1], (C[5]||i||ID[a]||ID[b]||GˆE[a,0] ||GˆE[b,i] ||T)), RVrf(GˆI[a], GˆI[b], GˆE[b,i], (C[5]||i||ID[a]||ID[b]||GˆE[a,0] ||GˆE[b,i] ||T), P)==true,

Now, Bob can use S and GˆE[a,1], Q[b,j] to initialize Double Ratchet, decrypt the payload and subsequent messages.

Initialize the Double Ratchet with the key material obtained from the handshake

Unlike OTRv3, the core iteration of Double Ratchet is asymmetric. The way to maintaining a normal Double Ratchet communication shall meet the following requirements:

1. The root key is numbered according to the number of iterations (executing RKCC() operation), and the numbers of the two sides always differ by plus or minus 1.
2. Each message contains the public part of the pre key in the sender chain.
3. If the root key number of the message sender is larger, the other party will iterate the root key **twice** after receiving the message (then the other party's root key number will be greater than itself), which are used to reset the receiver chain and sender respectively chain.
4. If the root key number of the message sender is small, the root key of the other party will not be changed, but the sender chain and receiver chain will continue to be iterated.

Therefore, the Double Ratchet protocol must be initialized in an asymmetric manner after the handshake: After the handshake, Alice holds: S, E[a][0], GˆE[b][0]; Bob holds: S, E[b][0], GˆE[a][0] .

Alice: E[a][1]=Rng(), (r[0], k[0]) = SCDK(S), (r[1], k[1]) = RKCC(r[0], Gˆ(E[a][1]E[b][0])), sender chain sc[a][0]{s, c} = (E[a][1], k[1]), receiver chain rc[a][0]{p, c} = (GˆE[b][0], k[0]).

That is, Alice needs to iterate the root key **one** more time.

Bob: (r[0], k[0]) = SCDK(S), sender chain sc[b][0]{s, c} = (E[b][0], k[0]), the receiver chain is not generated immediately.

After receiving the message, if it is found that the pre key GˆE in the message is not equal to any pre key recorded in the receiver chain, the party will

(r[n+1], k[n+1]) = RKCC(r[n], Gˆ(E*sc.s)), generate a new pre key E'=Rng(), (r[n+2], k[n+2]) = RKCC(r[n+1], Gˆ(EE')), sc = (E', k[n+2]), rc = (GˆE, k[n+1]).

If Alice sends the message first, because Bob has not initialized the receiver chain, it will trigger bob to iterate their root key twice to update the chain key; if Bob sends the message first, because their sender chain corresponds to the receiver chain of Alice, only the chain key is iterated.

We can find that after Double Ratchet is initialized in this way, any party sending a message first will not destroy the consistency of the context, so in theory, after IDAKE, either party can choose to iterate the root key one more time when initializing Double Ratchet. However, for security reason, the chain key should be updated in time to avoid excessive damage to the backward secrecy. Therefore, in Axolotl, we chose to let Alice who sent the first message iterate the root key one more time. Our ODAKE inherited this philosophy. At the same time, we also choose to let the Alice that initiated the IDAKE as the party to iterate the root key one more time.

UKS issue

A paper pointed out that TextSecure, the predecessor of Signal, allows an "Unknown key share" attack, that

is, Alice helps Eve and Bob establish a conversation, but makes Bob think he is in a conversation with Alice. But the "attack" in the original text is based on extremely loose management of public key fingerprints. If the other party already trusts one of our public keys, sending a new public key to the other party through the session established by the trusted public key shall be regarded as a guarantee action equivalent to issuing a certificate. On this basis, not only is Eve's public key regarded as Alice's own public key to be "guaranteed" to Bob, but also allowed Eve to use alice's own account, which is equivalent to giving Eve the opportunity to replace alice's identity. This approach is more to let Eve attack Alice theirself than an attack to Bob. Therefore, we only consider whether Alice might carry out such an attack without guaranteeing Eve's public key.

In fact, the aforementioned attacks in dakez-18 against OTRv3 and Axolotl's lack of online deniability can indeed be regarded as a UKS attack carried out without guaranteeing the public key of others, but it is very difficult in practice, either requiring Eve hands the intermediate result over to Alice for signature, and Eve uses the signed result to complete the handshake, or Eve should ask Alice to forward messages to Bob for her. The former approach places smaller burden on Alice, and Alice does not need to intervene in the conversation after generating the signature, but it is only suitable for offline handshake scenarios (for simplicity, assume that Alice and Eve hold and trust each other's public keys): Eve directly Establish a conversation with Bob through ODAKE, but with the help of Alice, pretend to be Alice, Eve needs to assemble the ring signature payload:

Payload = (C[5]||i||ID[a]||ID[b]||G ^ E[a,0]||G ^ E[b,i]||KDF(C[3], G ^ I[a]||G ^ I[b]))

and send it to Alice, and Alice sends her own public key GˆI[a], and signature result P=Rsig(I[a], GˆI[a], GˆI[b], GˆE[b,i], Payload) back to Eve so that she can complete the handshake. However, the result of such a lot of trouble is only that Eve can send offline messages to Bob in the name of Alice. If Bob decides to re-handshake with IDAKE or ODAKE, Eve, because of not holding I[a], is either unable to complete IDAKE thus exposed, or Bob's reply could not be decrypted because the ODAKE could not be completed. From this point of view, as long as the policy prohibits the conversion of offline sessions being converted to online sessions, although UKS cannot avoid it, the harm can be minimized.

Improvement of stage 2

Adaptation to XMPP

vault1317/signal-dakez can theoretically support any federated protocol. We choose XMPP for a trade-off based on its high extensibility and simple interface, and welcome new attempts from communities.

Management of sessions' life cycle

The original Axolotl saves the encryption session context to the secondary storage every time it changes, while in improvement of this stage, we implemented retain the context within RAM. Unless a special "commit" operation is carried out (it is basically never used), the context will never be saved to the secondary storage. In this way, the encryption session context will, of course, be destroyed when the program exited, which may reduce the potential leakage of secret data via the secondary storage to the least.

Hide real client id

As described above, instance ids capable to permanently identify a client instance should be hidden hidden in encrypted channels, but Axolotl needs a plain "id" to distinguish between different instances before decrypting a message. Currently this problem is solved as follows: Except for the real device id created during initialization of an instance, each instance will randomly generate its temporary device id during its start. After this, whenever a device id should present as plain text, the temporary device id is used. The real device id is encrypted and transferred to the peer during handshaking and the correspondence with the temporary device id is built. In afterward communications, clients will use these correspondences to translate a temporary device id to the real one, and find corresponding session for cryptographic operations.

The life cycle of a temporary device id

A temporary device id is generated during the start of an instance. If the instance never publishes a pre key bundle, the temporary device id is naturally destroyed when the client is stopped, but if the instance publishes a bundle, the temporary device is added into the device list of omemo and saved to the secondary storage. Until this instance goes online again, and all offline messages sent to the cached temporary device id and cached in the server is received, the temporary device will be destroyed and removed from device list when the instance receives its first online message. (indicating that all offline messages is completely received)

Management of pre key bundles

Differnet than the original axolotl/omemo, this protocol excludes any automatic action that will publish or fetch pre key bundles, instead, dedicated commands to publish and fetch a bundle are used. A pre key bundle is published, and another user's bundle is fetched for offline handshaking only when a user actively call these commands.

Integration with omemo

The original omemo is an XMPP Extension Protocol (XEP) that makes use of the pub-sub functionality to publish pre key bundles and automatically get their changes for the offline handshaking of Axolotl. vault1317/signal-dakez uses the main consistent message format with omemo, but there are changes in:

- Definition of own dedicated namespace.
- As mentioned above, all "device id" present as plain text in messages, device lists, and pre key bundles is the temporary device id.
- A pre key bundle does not contain an identity key any more.
- When the saved temporary device id is destroyed as mentioned above, pre key bundles published on the server in its name will be retracted with the pub-sub functionality of XMPP. Apart from such auto-retracting mechanism, a user can also manually delete a saved temporary device id and retract corresponding bundle, and even retrach all temporary device id (delete the whole device list) and bundle once published by the account.
- Idake is accomplished with omemo empty messages with idake handshaking messages written into the "key" node. However, because the peer's temporary device id has not been known before idake handshaking, client should write a dedicated sequence encoded with base64 to the key" node, with "rid" attribute set to 0. The peer can get our temporary device id from the "sid" attribute of the "header" node after they receive the message and recognize the sequence, and they will set our temporary device id to the "rid" attribute of "key" node, write real idake handshaking message to the data of "key" node in order to start the real idake handshaking.
- A user can choose to actively terminate their encrypted session, with local corresponding encrypted session context destroyed. If the user is online, an "termination notification" will be encrypted and sent as body text to the remote peer of the session to be terminated. The remote peer will also terminate the session and destroy the context after they receice and decrypt the message containing the "termination notification".

Future

Although the vault1317/signal-dakez protocol does everything in its power to achieve deniability and public key concealing (key exchange without revealing the identity/fingerprint). It doesn't make sense to use the machine in our daily life if its privacy can't not be protected. Cypherpunk believes that this is the only way to protect the privacy for our machines. The current version still have some unfinished features:

Stage 1

1. (already finished in stage 2) The current version has not yet completely concealed the client ID (device ID) in an encrypted session because Axolotl requires the client to identify the instance of the client from which the ciphertext originates before the encryption. However, in fact, at this stage, we only need to distinguish instances from each other so that we can define "external id" and "internal id" with different functions, where the internal id is actually corresponding to the public key and changed along with the public key, and "external id" resembles an OTR instance tag that is generated randomly each time the client logs on, provided that multiple clients online at the same time has different "external id". The specific design and connection with ODAKE will be completed in the future.
2. (already finished in stage 2) The current version does not have any modifications in the Double Ratchet scheme, that is, it will be saved in the secondary storage every time the session context changes, e.g: sqlite database update to comply with internal API of libsignal-protocol-c. In future versions, a mechanism will be introduced that does not save the context of the session, but only saves the necessary parameters and disables the shift between offline and online sessions.
3. The long-term version will follow OTR, which is convenient for forging a message and provides tools for counterfeiting by utilizing symmetric encryption mode.
4. (already finished in stage 2, adapted to XMPP) The current version only demonstrated the vault1317/signal-dakez protocol based on UNIX domain socket. The real-life implementation will support mature and popular communication protocols. As far as we can tell is the protocol is the one which must support federation.

Stage 2

1. Design a dedicated handshaking method for group chatting. This method should word without expose one's real JID to the chatting room.

Gratitude

Thanks for the work of cypherpunk and 0ldsk00l hackers in past 40 years, specially the free software projects that use cryptography to protect users's security and privacy. Special thanks to the project and papers this project relys on which is very very enlightening for all of us.

- OTRv3
- OTRv4
- Axolotl a.k.a "Signal Protocol"
- libsignal-protocol-c
- protobuf-c
- OpenPGP
- olm
- OMEMO
- Proteus
- OTRv4
- axc
- libomemo
- lurch
- libgcrypt
- libevent
- GNU Readline
- Improved Strongly Deniable Authenticated KeyExchanges for Secure Messaging
- How Secure is TextSecure?
- A Formal Security Analysis of the Signal Messaging Protoco
- The X3DH Key Agreement Protocol
- The Double Ratchet Algorithm

- The XEdDSA and VXEdDSA Signature Schemes
- Deniable Authentication and Key Exchange
- Deniable Key Exchanges for Secure Messaging
- One-round Strongly Secure Key Exchange withPerfect Forward Secrecy and Deniability
- Ed25519 to Curve25519
- May the Fourth Be With You: A Microarchitectural Side ChannelAttack on Several Real-World Applications of Curve25519
- Curve25519:newDie-Hellmanspeedrecords