Wickr Messaging Protocol

TECHNICAL PAPER

Authors

Chris Howell, Tom Leavy & Joël Alwen

Special thanks to Whitfield Diffie, Paul Kocher, Dan Kaminsky, Adam Shostack, Scott Stender & Jesse Burns for reviewing this paper and/or code and providing their insightful comments and invaluable advice.

Table of Contents

Introduction

The Wickr Secure Messaging Protocol provides a platform for secure communications. It is a method for sending messages with a set of security properties that we will explore in what follows.

This document is intended as a summary of the protocol and an aid to those who wish to audit the source code. Our goal is to offer enough technical detail to allow security experts and cryptographers to observe the protocol's security design, use of cryptographic primitives, etc., while also providing value to a wider audience of users and interested parties. Full technical detail can be obtained by reviewing the source code, which is available for review here.

Protocol Objectives

The primary objective of the Wickr Secure Messaging Protocol is to provide secure communication between two or more correspondents. There are a variety of properties that can be demanded in order to call a system "secure". At minimum it means that the system provides authenticity and confidentiality: no unauthorized party can inject a message into the system and no unintended party can get to understand the communications without being given them by one of the correspondents.

Opposition to this objective may come from a variety of directions. The most common application of the concept of opponent, will be to other Wickr users. A user wants to share some information with some users, other information with other users, and perhaps some with nobody. Other users may reasonably or unreasonably be interested in acquiring information whether its owner wanted to share it with them or not. Generally, other Wickr users are the most common form of opponent but the least powerful. By definition their actions are taken through Wickr systems and thus subject to some degree of control by Wickr.

The next most numerous class of opponents are probably system penetrators, people who attack Wickr servers, or perhaps even the users themselves, by communications that do not pass primarily through Wickr systems. Attacks of this kind have been common for a decade

and are likely to continue. Wickr counters these attacks with operating system security, firewalls, and other measures, but is ultimately aware that such techniques have failed in the past and may at some point fail again. Therefore Wickr also attempts to avoid knowing anything it does not need, to carry out its operations.

The most threatening opponents are those who can take physical control of Wickr's systems and perhaps demand the cooperation and assistance of Wickr personnel. Such opponents might conceivably be overt criminals who break into a Wickr server room and perhaps take the staff hostage but they are more likely to be police armed with a court order requiring staff assistance. It is in this case that Wickr's architectural security approach comes into its own. Wickr cannot reveal what it doesn't know; this is the strength of Wickr's "ignorance by design" and the foundational principle from which the protocol was designed.

Protocol Overview

A faithful implementation of the Wickr Secure Messaging Protocol provides the following services:

- **End to End Encryption** – Messages and encryption keys are available only with in Wickr applications and are not disclosed to network attackers or Wickr server operators.
- **Perfect Forward Secrec**y – Old message content is not compromised if the long-term key of a user or device is compromised. Backward secrecy is also provided against passive adversaries.
- **Protection of Metadata** - Node identity information stored in message head ers is hidden from attackers who lack message context, and source information.

The primary cryptographic actor in the protocol is the node. Concrete representations of nodes are Wickr applications, or "apps", which can also be referred to as devices when installed on devices such as mobile phones. Nodes perform all message encryption and decryption functions. Nodes also create, sign, and publish public asymmetric key components

to facilitate secure messaging operations and validate signatures of keys received from other nodes.

Users serve as top-level identities in the messaging system (e.g. 'Alice'). Users are roots of trust from a cryptographic perspective, and hold the top-level signing keys that tie nodes to user identities. The protocol's ability to maintain this trust chain from node to user is what affords Wickr users great flexibility to securely add devices, or run on several devices at once.

At the highest level of the protocol view, nodes encrypt messages with a strong symmetric cipher using randomly generated keys. Nodes pass the random symmetric keys ***in the message*** to recipient nodes using strong asymmetric cryptography. To decrypt messages, receiving nodes reverse the process, using asymmetric cryptography to extract the random keys, and the random keys to decrypt the ciphertext. Group messaging is supported in the same manner by encrypting a message, encrypting the key to that message multiple times (for multiple nodes), bundling, and sending a single message to multiple nodes.

Pools of public asymmetric key components, or ***ephemeral keys***, are maintained for messaging operations. This ensures that strong forward secrecy is maintained in both synchronous and asynchronous messaging environments. These keys are signed and validated up through the root of trust on every use. Private key components never leave the device on which they are generated, ensuring that none other than sender-designated recipient node(s) can decrypt messages.

This design makes the protocol resistant to message authenticity and confidentiality threats posed by various actors along the entire path of delivery from device to device, including those posed by a rogue server.

The remainder of this document describes the protocol at length. To review the source code please see [wickr-crypto-c.]

Cryptographic Operations

The Wickr Secure Messaging Protocol relies on the following classes of cryptographic primitives.

- **DH(K1, PK2)**

  Calculate shared secret using Non-Interactive Key Exchange

  The current implementation uses Elliptic Curve Diffie Hellman Key Exchange with P521 key pairs.

- **Sk(Data)**

  Calculate cryptographic signature of data using private key k

  The current implementation uses Elliptic Curve Digital Signature Algorithm with P521 key pairs.

- **Ek(Data)**

  Encrypt data with symmetric cipher using k

  The current implementation uses AES 256 in GCM mode unless other wise specified.

- **Dk(Data)**

  Decrypt data with symmetric cipher using k

  The current implementation uses AES 256 in GCM mode unless other wise specified.

- KDF(Data)

  Derive key from data

The current implementation uses HKDF with SHA-256 as the underlying hash function or Scrypt from low entropy sources unless otherwise specified.

These primitives operate on several keys and identifiers, categorized as follows:

Identity Keys – Long-lived asymmetric key pairs provide for the identity of participants in the protocol.

Ephemeral Keys – Short-lived asymmetric key pairs used to exchange a shared secret between participants in the protocol before being discarded.

Symmetric Keys - Symmetric keys used to directly encrypt data or are a significant portion of the material from which an encryption key is derived. These keys are generated from platform- APIs for gathering random numbers suitable for cryptographic operations.

Identifiers – Unique numbers generated or gathered from users and/or devices, typically in the form of SHA-256 hash values. These are sometimes used as optional entropy or material to support obfuscation.

Wickr's implementation of the protocol is designed to accommodate different cryptographic primitives should advancements in cryptography require they be changed.

User and Device Enrollment

Wickr's security begins with enrollment of a user and their first device. This configures the root keys used to secure node-to-node communications that follow.

User Enrollment

Users enroll using the Wickr application on their own device. The following items critical to the protocol are created for each Wickr user during enrollment:

Identity Keys

- **Kr and PKr**

  The Root Identity Key Pair for the user

  The root of trust for a Wickr user account. Used to sign Node Identity Public Keys.

Symmetric Keys

- **Krs**

  The Remote Storage Root Key

  Randomly generated key used to encrypt account-level backups stored on Wickr servers.

- **Knsr**

  The Node Storage Root Key

  Randomly generated key material used to derive the key used to encrypt data stored on a device.

- **Krbk**

  The Recovery Bundle Key

  Randomly generated key used to protect identity keys stored on Wickr servers.

Identifiers

- `IDr`

  `Root identifier`

  `Unique value to identify a user on the Wickr system, typically in the form of a SHA-256 hash value.`

`PKr` and `IDr` are stored on Wickr servers and provided to communication peers along with profile data. `Kr, Krs`, and `Knsr` provide critical identity and data protection services and are shared between devices. Current practices for protecting `Krbk` from a malicious server are described in Wickr Recovery Bundle, later in this document.

Device Enrollment

Device enrollment occurs any time a user logs in to Wickr using a new device and after user enrollment to establish the user's first device. The terms "device," "app," and "node" are used somewhat interchangeably in this document and all generally describe Wickr application instances from different perspectives. The terms "app" and "device" generally describe the application from the perspective of the system or physical device that hosts Wickr software, whereas "node" generally refers to the application in the context of an encryption scheme. The following items are created for each Device Enrollment:

Identity Keys

- `Kn and PKn`

  `The Node Identity Key Pair for the device`

  `The source and destination identity for messages. Used to sign message ciphertext.`

Symmetric Keys

- `Klsd`

  `The Local Storage Device Key`

  `Computed using KDF(Knsr || Device Data) . Used to protect data when stored on a device.`

Identifiers

- `IDn`

  `Node Identifier`

  `Randomly generated, but not used as a secret key.`

Successful device enrollment requires the `Kr, Krs`, and `Knsr` keys created during user enrollment as input.

The Wickr app calculates the signature of the `Node Identity Public Key` using the `Root Identity Private Key, SKr(PKn)` and calculates the `Local Storage Device Key, Klds = KDF(Knsr || Device Data)`. Device Data refers to device-specific data and/or identifiers derived from installed hardware or operating system sources that are unique, constant across application installs but not necessarily secret. The Wickr app stores `PKn` and `SKr(PKn)` within the user's profile on Wickr servers.

Message Exchange

The Wickr Secure Message Protocol can be better understood as providing node-to-node communication rather than user-to-user communication. A Wickr user can have multiple associated devices, each of which is considered a node in the protocol.

Consider a user with two devices who sends a message to a user with three devices. In this case, the initiating node sends an encrypted message to four nodes – the three device nodes associated with the other user, and the sending user's own second device node. The notion of user identity is important – device nodes' public identity keys are verified as being trusted by the target user identity – but a node and its associated identity keys are the primary actors in this protocol.

Ephemeral Key Pairs

Each Wickr device node creates and refreshes a pool of asymmetric key pairs to be used for Diffie-Hellman key exchange when receiving messages, referred to as `KEn` and `PKEn`. The public components of these key pairs are signed by the originating device's Node Identity Private Key. This signature, `SKn(PKEn)`, is uploaded to Wickr servers along with `PKEn` and a unique identifier for the key, `IDken`.

Sending a Message

When a user wishes to send a message to fellow users, the Wickr app performs the following steps:

Prepare Key Exchange Data

1. Retrieve the receiving users' profile data, including each user's Root Identity Public Key PKr, a list of that user's associated device nodes, and the Signed Node Identity Public Key (SKr(PKn), PKn), for each node.
2. Build a list of recipient nodes from the union of those device nodes retrieved in Step 1 and the sending user's own device nodes.
3. Retrieve the signed public components of an Ephemeral Key Pair (SKn(PKEn),PKEn), and associated Key ID IDken from the server for each recipient node.
4. Validate the signature chain for each retrieved Ephemeral Key Pair:

a. PKEn corresponds to a valid SKn(PKEn)

b. PKn corresponds to a valid SKr(PKn)

c. PKr corresponds to the expected user identity

Prepare Packet Header

1. Generate a random message payload encryption key: Kpayload
2. Calculate packet header encryption key from sender profile information: Kheader = KDF(IDr_s || IDn_s). Note that packet header encryption is only intended to protect the information from attackers who lack knowledge of the sender of the message.

3. Generate an Ephemeral Key Pair for the sending node: KEs and PKEs
4. Calculate a unique exchange key for each recipient node, using the identifier of the receiving node and the sender and receiver's Root Identity Public Key: Kexchn=KDF(DH(KEs, PKEn) || PKr_s || PKr_r || IDn)
5. Create Key Exchange Data for each recipient node: KEDn = EKexchn(Kpayload) || IDn || IDken
6. Create Key Exchange List: KEL = KED0 || KED1 || … || KEDn-1 for n recipient nodes
7. Create Encrypted Packet Header: EPH = EKheader(PKEs || KEL)

Prepare Packet Content

1. Create message metadata including content type and ephemerality configuration
2. Create encrypted message payload: EP = EKpayload(Message Metadata || Message Content)
3. Create packet signature: PS = SKn(EPH || EP)
4. Create serialized packet: P = Version || Cryptographic Configuration || EP || EPH || PS

The serialized packet is then sent to Wickr servers, which then forward the packet to recipient nodes.

Receiving a Message

A recipient node is responsible for identifying the appropriate key material to decrypt the message content. To this end, it receives important information from the Wickr server– the identifier of the sending node, IDn_s the identifier of that node's root IDr_s, and user profile information.

This information is combined with device state to perform the following steps:

1. De-serialize packet and set appropriate version and cryptographic configuration
2. Verify packet signature:

a. (EPH || EP) corresponds to a valid SKn(EPH || EP)

b. PKn corresponds to a valid SKr(PKn)

c. PKr corresponds to the expected user identity

3. Calculate packet header encryption key: Kheader = KDF(IDr_s || IDn_s)

4. Decrypt packet header: DKheader(EPH) = PKEs || KEL

5. Retrieve own KED (Key Exchange Data) from decrypted KEL (Key Exchange List) using own node identifier for lookup. KEDn = EKexchn(Kpayload) || IDn || IDken

6. Use IDken to identify the ephemeral key pair selected by the sending node for key exchange and retrieve from local storage: KEn, PKEn

7. Calculate exchange key shared with sending node: Kexchn= KDF(DH(KEn, PKEn) || PKr_s || PKr_r || IDn)

8. Calculate DKexchn(EKexchn(Kpayload)) to retrieve Kpayload.

9. Decrypt message: DKpayload(EP) = Message Metadata || Message Payload

The message payload is stored in local storage encrypted with the Local Storage Device Key, `Klds`. All short-lived keys are deleted shortly after. The Wickr app will carry out actions in accordance with the message metadata, including deleting the message after its Time to Live has expired.

Protocol Extensions

The Wickr Secure Messaging Protocol provides a platform for secure communications. In the simple case, a message payload contains only a text message. However, message metadata can indicate that other materials are to be exchanged, in which case the message payload includes one or more shared secrets used to protect those materials. This general design permits the Wickr Secure Messaging Protocol to support end-to-end encryption for file transfer, audio/video communications, or future use cases.

Wickr Security Practices

The Wickr Secure Messaging Protocol is the foundation for Wickr security, which is provided through the implementation of many security controls and practices in our products, the full breadth and description of which are beyond the scope of this document. The following items are noted as a sampling of security practices supporting the current implementation.

Ephemeral Messaging

Each encrypted message is associated with an expiration time, after which time apps must delete the message.

Ephemeral Key Management

Ephemeral keys are managed in pools to provide strong forward secrecy, even in cases where devices go offline for periods of time. The primary key pool is maintained on the Wickr server. Sending nodes deplete the pool by using keys; recipient nodes replenish it by publishing them. Online devices replenish the pool immediately. Offline devices replenish the pool as soon as they come online.

A corner case exists in that if the key pool is exhausted, the last key will need to be re-used until the pool is replenished, thus expanding the amount of material protected by a particular ephemeral key. What makes this a corner case is that in our implementation, pool size is dynamic depending on the needs of the device and sufficiently high to handle devices being offline for reasonable periods of time.

An attack scenario could be contrived in which an attacker purposefully depletes a device's key pool in an attempt to undermine forward secrecy. Even if an attacker could deplete the pool, exploiting the condition would be highly difficult, requiring the attacker to compromise the victim's device and prevent the victim from coming online to replenish the pool. However difficult, this attack is mitigated in our implementation through the use of secondary key pools that are maintained offline within the apps and not subject to depletion by arbitrary attackers.

Peer Identity Management

The Wickr Secure Message Protocol specifies that a chain of trust exist between a user's `Root Identity Key Pair,` their devices' `Node Identity Key Pairs`, and each Nodes' `Ephemeral Key Pairs`. This chain is validated by message recipients to ensure that messages are received from the expected user.

The protocol does not specify how a user's identity is associated with their `Root Identity Key Pair`. The Wickr app provides multiple means to associate a Root Identity Public Key with a human user. Once established, this association is saved, or "pinned," on each of a user's devices. If an attacker poses as a communication peer but has a different Root Identity Public Key, this communication will be flagged as not authenticated.

Wickr Recovery Bundle

Three important keys, `Kr, Krs`, and `Knsr` are created during User Enrollment and must be shared securely between devices. This is accomplished by encrypting and storing them on Wickr servers.

This recovery bundle is encrypted using `AES 256 in GCM` mode with `Krbk` prior to storage. `Krbk` is then encrypted with a key derived from the user's passphrase using scrypt and stored along with the recovery bundle. This method of storing `Krbk` adds an important level of usability to the platform to allow the core of the user's identity to be transferred between devices using only the user's passphrase. We recognize that this convenience does not come without risk, but with scrypt and other platform countermeasures in place we believe we've reduced it as much as possible.

The protocol also supports hardened key scenarios. One is to require the user to maintain `Krbk` in their own records and provide it to a device during enrollment. In this case, the encrypted recovery bundle would be maintained on Wickr servers but not the passphrase-encrypted `Krbk`. Similarly, `Kr, Krs,` and `Knsr` can be maintained offline and used only during scenarios that require access to those keys.

Encrypted Local Storage

The Wickr app creates an encrypted storage container on each device to store sensitive data

such as identity keys, messages, and account data. This container is decrypted during active logon sessions and its contents used for normal operation. When the user logs off, the container is encrypted with `Klds`, and this key is removed from local and persistent memory.

`Klds` is stored in an encrypted format so that it may be recovered upon the next user login. The key used to encrypt and decrypt `Klds` is derived from the user's passphrase using scrypt. Successful login in this case is tantamount to being able to successfully decrypt `Klds` and access encrypted storage material. Those users who wish to always remain logged in to Wickr simply store the password-derived key in platform-provided secure storage. In this way, sensitive material is always encrypted when the Wickr app is not active.

Secure Transport

The Wickr Secure Messaging Protocol is designed to protect message content even if the data described above were transmitted in cleartext over an unprotected network. However, traffic analysis would reveal modestly sensitive information, such as identifiers for sending and receiving nodes and general traffic patterns.

Wickr protects the protocol with two additional layers of security. First, app-server requests and responses are encrypted with a rotating shared secret key using `AES 256 in CFB` mode. Second, the Wickr app tunnels this `AES encrypted data inside of TLS.` These protections provide redundant defense-in-depth measures that protect message metadata as well as content.

Conclusion

The Wickr Secure Messaging Protocol is designed to provide end-to-end encrypted communications. It accomplishes this through the use of standard cryptographic primitives which ensure the confidentiality and integrity of messages while in transit and while stored on Wickr servers. Wickr believes this protocol achieves our goal of keeping our customers in control of their content, and we welcome your feedback to strengthen it further.

The source code for the Wickr Messaging protocol is available here:

https://github.com/WickrInc/wickr-crypto-c.

[1] The algorithms in use as of December, 2016 are noted.

[2] The notation '||' indicates the concatenation of values, e.g. 'a || b' means the concatenation of a and b, where the encodings of a and b include type and length as appropriate.