Kintsugi: Decentralized E2EE Key Recovery

Emilie Ma[1][0009−0005−3322−0805] and Martin Kleppmann[2][0000−0001−7252−6958]

[1] University of British Columbia, British Columbia, Canada
contact@emilie.ma
[2] University of Cambridge, Cambridge, United Kingdom
martin.kleppmann@cst.cam.ac.uk

Abstract. Kintsugi is a protocol for key recovery, allowing a user to regain access to end-to-end encrypted data after they have lost their device, but still have their (potentially low-entropy) password. Existing E2EE key recovery methods, such as those deployed by Signal and WhatsApp, centralize trust by relying on servers administered by a single provider. Kintsugi is decentralized, distributing trust over multiple recovery nodes, which could be servers run by independent parties, or end user devices in a peer-to-peer setting. To recover a user's keys, a threshold $t+1$ of recovery nodes must assist the user in decrypting a shared backup. Kintsugi is password-authenticated and protects against offline brute-force password guessing without requiring any specialized secure hardware. Kintsugi can tolerate up to t honest-but-curious colluding recovery nodes, as well as $n-t-1$ offline nodes, and operates safely in an asynchronous network model where messages can be arbitrarily delayed.

Keywords: Decentralized key recovery · End-to-end encryption.

1 Introduction

As end-to-end encrypted (E2EE) services gain popularity, an important problem is account or key recovery. With non-E2EE services, a user who loses their device or switches to a new one can simply log in on a different machine with their username and password. In an E2EE setting, this is problematic: the password may lack sufficient entropy, so it cannot securely seed a key for decrypting a backup of the user's data. Common recovery mechanisms for E2EE platforms include user-selected recovery passwords [26], recovery codes [27,20,18], short PINs with hardware-enforced guess limits [11], local copies of recovery files [4], or a designated recovery contact [8,4,22]. However, a key issue with existing recovery methods is the trend towards centralization: users cannot verify that the central service implements recovery securely or reliably. Existing mechanisms have associated tradeoffs:

– Signal's Secure Value Recovery [11] and WhatsApp's E2EE backups [26] require the user to remember a four-digit PIN and rely on trusted hardware, like hardware security modules (HSMs), to limit PIN guesses. In the case of Signal, multiple hardware elements are distributed across several locations

and cloud providers. Nevertheless, the hardware on both platforms remains a centralized system, operated by a single party that must be trusted to correctly manage the key recovery infrastructure.
- Apple iCloud allows users to designate a single recovery contact [8]. If this contact is untrustworthy and has physical access to one of the user's devices, they are able to take over the user's account.
- Recovery codes used by services like MEGA [20] or LastPass [18], and Bitcoin wallet recovery seed phrases consisting of twelve random words [21], avoid trusting any other parties. However, their high-entropy nature makes them impractical to remember, and paper backups are prone to being lost. Digital recovery files are at risk of device loss or failure or improper storage in an unencrypted cloud service, undermining the system's E2EE properties [10].
- Applying Shamir Secret Sharing [23] to split a recovery key across multiple contacts raises the issue of authentication: when a contact receives a request to participate in secret reconstruction, the contact needs to decide if the request is genuine. This is susceptible to social engineering, tricking contacts into revealing their secret shares to an adversary. There is also no protection against a threshold of contacts colluding to reconstruct the key directly and no way to change the set of contacts without repeating the secret-sharing process.

Centralized infrastructure is a risk for applications that require metadata privacy (e.g. anonymity networks) or where the infrastructure may be shut down outside of their control (e.g. hosting for government-sanctioned activists), besides requiring users to blindly trust the service. For example, WhatsApp uses a HSM-based Backup Key Vault service to protect against brute-force attacks, but this requires trusting the HSMs, which are under WhatsApp's sole control [16].

There is also an aspect of cost: Signal, WhatsApp, and Apple iCloud all rely on HSMs to ensure only authenticated users can recover their data. However, these HSMs are expensive and difficult to deploy. For instance, Signal's staging (non-production) Secure Value Recovery implementation costs $2,100/month to run [11]. In turn, this limits decentralization efforts: while volunteers may be happy to help run nodes for a decentralized service on consumer-level hardware (e.g. Tor node operators), it becomes cost-prohibitive to participate if specialized hardware is required. We thus wished to explore a new space in recovery protocols that avoids any reliance on centralized infrastructure or specialized hardware.

In this paper, we propose Kintsugi, a decentralized key recovery protocol based on a peer-to-peer network of recovery nodes. Nodes may be servers operated by different parties, or end-user devices of contacts, or a mixture of the two. To recover their secret key on a new device, the user must keep track of and provide a password. Then, the device communicates with at least $t + 1$ recovery nodes, where t is the reconstruction threshold chosen by the user when they set up their recovery contacts. The recovery nodes do not need to check that requests are authentic; they only need to rate-limit requests to prevent online brute-force attacks on the user's password. Kintsugi is capable of handling up to t honest-but-curious nodes who participate in the protocol correctly but may

collude in trying to compromise the user's secrets, and $n - t - 1$ offline nodes, where n is the total number of recovery nodes for this user. To obtain the user's secret key, $t+1$ recovery nodes would need to collude and additionally mount an offline brute-force attack on the user's password. Users can update their recovery nodes at any time, and even if some recovery nodes are compromised over time, the user's account remains secure thanks to secret refreshing, which regularly regenerates the shares of secrets held by the nodes without changing the joint secret itself. Kintsugi can also operate in an asynchronous network model, where messages may be arbitrarily delayed.

2 Threat Model

We assume that all recovery nodes correctly follow the protocol (i.e. are not Byzantine). Byzantine fault-tolerance is not yet implemented in our prototype, but could be added based on prior work [29]. Each user can choose the number of recovery nodes they use, n, and the key reconstruction threshold, t. Kintsugi can tolerate up to t honest-but-curious recovery nodes, which may collude, attempt to guess the user's password, and do not rate-limit responses, but otherwise correctly follow the protocol, and $n - t - 1$ offline nodes, which do not respond to recovery requests.

We assume an adversary who can both passively eavesdrop and actively interfere with network traffic, including making recovery requests to recovery nodes. Additionally, the adversary can obtain the secret shares of up to t recovery nodes (the same t honest-but-curious nodes previously mentioned) within any given refresh interval and perform brute-force attacks, although we assume they cannot compute discrete logarithms. Alternatively, we also allow an adversary who obtains more than t secret shares, but who lacks computational resources such that offline brute-force attacks have a negligible success rate.

Finally, we assume an asynchronous network model, where messages may be arbitrarily delayed.

3 Kintsugi Protocol

Several properties are required for Kintsugi's design.

- Recovery of the user's key requires the participation of a group of recovery nodes. An adversary wanting to reconstruct the user's key should require both collusion among at least $t+1$ recovery nodes and brute-force password guessing. We accomplished this via an threshold Oblivious Pseudo-Random Function (OPRF) exchange. See section 3.1 for more details. This OPRF usage was inspired by the OPAQUE protocol [28].
- A threshold of recovery nodes must be involved in the key reconstruction. This requires a secret sharing scheme, like Shamir's Secret Sharing [23].

- Users must be able to change their recovery nodes and the threshold of nodes required to recover the user's key. This must be possible at any time, not requiring a delay until the start of a next epoch. Former recovery nodes should also not be able to participate in the reconstruction of the user's key. This requires a *dynamic, proactive* secret sharing scheme: the set of recovery nodes can be updated (dynamic committee) and the shares held by recovery nodes can be renewed while keeping the joint secret the same (proactive refresh). Using a proactive secret sharing scheme also guards against shares being compromised over time, since old shares will be invalidated on each refresh.
- Kintsugi should continue to correctly operate in an asynchronous network model where messages can be arbitrarily delayed, as we reasonably expect recovery nodes to go offline (e.g. maintenance, server outage, network interruption). Synchronous protocols become unsafe in an asynchronous network, because the nodes whose messages are delayed may be ejected, causing fault-tolerance against honest-but-curious or genuinely offline nodes to decrease.

Kintsugi consists of three protocols: user registration, key recovery, and changing recovery nodes. A prototype implementation, built with Rust, Tauri, and libp2p, is available as open source[3].

3.1 Threshold OPRF

We combine an Oblivious Pseudo-Random Function (OPRF) with Shamir secret sharing (SSS) [23] to obtain a threshold OPRF [14], which is performed once per user registration or key recovery request. This OPRF output is used as an encryption key for the recovery data backup.

Consider an elliptic curve group $\mathbb{E}$ of prime order q. We use the Ristretto curve [25], a modification of Curve25519 to eliminate cofactors, because of its efficiency and its compatibility with off-the-shelf OPRF libraries. Let pwd be the user's password. Kintsugi's OPRF is a deterministic function $f(P, s)$ where:

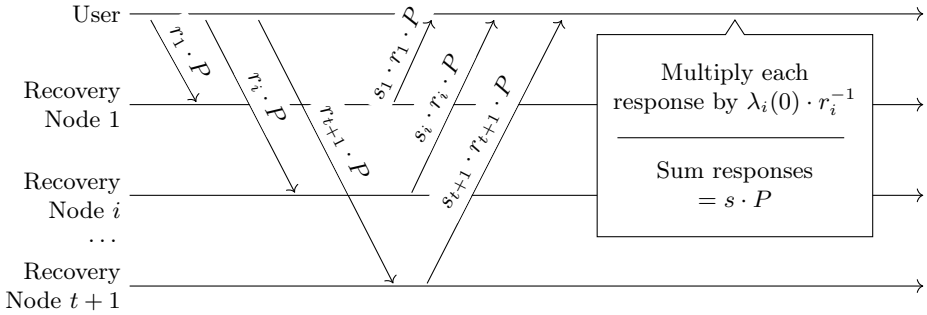
- $P = \text{HashToCurve}(pwd) \in \mathbb{E}$, the user's password hashed to a point on the curve using a scheme like Elligator [9]
- $s \in \mathbb{Z}_q$ is a secret selected by the user's device during registration, of which each recovery node j holds a SSS share s_j
- the user inputs P to the function f without knowing s, and learns the output $f(P, s) = s \cdot P$, the scalar product in the elliptic curve group
- the recovery node j inputs s_j to the function, and learns neither P nor $s \cdot P$
- the output $f(P, s) = s \cdot P$ is computationally indistinguishable from random

To perform an OPRF evaluation, the user's device first generates the uniformly random secret $s \in \mathbb{Z}_q$ and splits s into SSS shares that are distributed to the recovery nodes. SSS works by defining a threshold t and a polynomial $SSS(x) = s + c_1 x + c_2 x^2 \dots c_t x^t$, where the coefficients $c_i \in \mathbb{Z}_q$ are sampled

[3] https://github.com/kewbish/kintsugi

uniformly randomly. For each user, each recovery node is assigned an integer index $i = 1, 2, 3 \ldots$, and its secret share $s_i = SSS(i)$ (i.e. the polynomial evaluated at its index using arithmetic in the field $\mathbb{Z}_q$). Any collection of $t + 1$ points can then be interpolated together to recover the secret constant term, which is $s = SSS(0)$.

Fig. 1. Threshold OPRF evaluation after secret share setup.



See Figure 1 for an example OPRF evaluation flow. The user's device chooses and stores a random blinding scalars $r_i \in \mathbb{Z}_q$ to prevent recovery nodes from learning P. The user's device sends $r_i \cdot P$ to every recovery node i, which multiplies it by s_i and returns the result. After the user's device has received $t + 1$ such results, it multiplies each response $s_i \cdot r_i \cdot P$ by r_i^{-1} to obtain $s_i \cdot P$. It then applies Lagrange interpolation to find $s \cdot P$.

Let $\lambda_i(x)$ be the Lagrange polynomial:

$$\lambda_i(x) = \prod_{\substack{1 \le m \le t+1 \\ m \ne i}} \frac{x - x_m}{x_i - x_m}$$

where x_k is the index at which the SSS polynomial was evaluated (i.e. the recovery node's index) for the kth secret share being interpolated.

Then, the Lagrange interpolation to reconstruct $s \cdot P$ is:

$$\begin{aligned} &\lambda_1(0) \cdot s_1 \cdot P + \lambda_2(0) \cdot s_2 \cdot P + \cdots + \lambda_{t+1}(0) \cdot s_{t+1} \cdot P \\ &= (\lambda_1(0) \cdot s_1 + \lambda_2(0) \cdot s_2 + \cdots + \lambda_{t+1}(0) \cdot s_{t+1}) \cdot P \\ &= s \cdot P \text{ (by Lagrange interpolation of the shares of } s) \end{aligned}$$

This computation is performed in the group $\mathbb{E}$, with $\cdot$ denoting scalar multiplication and $+$ denoting point addition. However, the result is the same as if the secret s had been reconstructed in $\mathbb{Z}_q$ field arithmetic, then multiplied with the point P: the interpolation is performed "in the exponent" [14].

The output $s \cdot P$ is the encryption key for the recovery data backup, as detailed below. Crucially, no party in this protocol can reconstruct s, because

each node only returns the scalar multiplication of their share with the user-provided password point, and never the actual share. Even if the point P is chosen by an adversary, they cannot compute s, as that would require computing a discrete log.

The protocol would be insecure if $P = p \cdot G$, where G is a generator of $\mathbb{E}$ and p is an efficiently computable function of *pwd*. The adversary could send G to the recovery nodes without knowing *pwd*, compute $s \cdot G$ from the responses, and then perform an offline brute-force attack on the password, trying many values p until the correct OPRF output $p \cdot s \cdot G = s \cdot P$ is returned. In contrast, if the discrete logarithm of P is not known, as is the case when using a scheme like Elligator [9] to hash *pwd* to a point on the curve, the adversary must send a separate request to the recovery nodes for each password attempt. This turns an offline brute-force attack into an online one, allowing recovery nodes to enforce rate-limiting on password guesses.

3.2 Registration and Key Recovery

During registration, the user provides a username and a password, and hashes their password into their secret curve point P. The user also samples a uniformly random secret $s \in \mathbb{Z}_q$, splits it into SSS shares, and distributes shares to their selected recovery nodes via encrypted and authenticated channels. The OPRF output $s \cdot P$ is then used as the key to encrypt a backup of the user's recovery key, and any other data they wish to restore after a device loss, with an authenticated encryption scheme. This encrypted backup is sent to all recovery nodes, which store it along with their secret share s_j.

When recovering their key, we assume users will not have access to or remember their set of recovery nodes. On registration, we use a distributed hash table (DHT), a decentralized, replicated key-value store, to store a mapping from username to recovery node identifiers.

In our implementation, we used the Kademlia DHT [19] provided by libp2p [2] to map usernames to a list of the recovery nodes' libp2p addresses. To prevent DHT poisoning or unauthorized updates, users provide their public key when first updating their DHT entry and sign subsequent updates. This approach is not confidential, leaking the identities of each user's chosen recovery nodes. This may be problematic for privacy if the recovery nodes are the user's contacts in a peer-to-peer setting, for example. However, the user would otherwise be required to keep track of their recovery nodes, and we argue that exposing this limited information is an acceptable tradeoff. We also note that Kademlia does not provide any integrity guarantees: this is not an issue given our threat model assumes no Byzantine nodes, but a misbehaving DHT node could otherwise serve fake recovery node information. Future implementations of Kintsugi may benefit from a different, BFT DHT or from storing recovery node information on a central server to simplify the threat model.

Users recover their key by looking up their recovery nodes by username in the DHT, inputting their password, and then initiating a threshold OPRF evaluation. This recovery can be performed on a different device than the device used

during user registration. The user also downloads their encrypted key backup from one of the recovery nodes. If the password was correct, they can decrypt the backup with the computed OPRF output $s \cdot P$, and thus recover their data. If decryption fails, this may indicate that the password was incorrect, or that one of the recovery node responses was corrupted or modified by an active network adversary. The only way of trying a new password is to restart the threshold OPRF evaluation with a new P. Recovery attempts may be replicated as a transparent audit log across nodes to alert users to potentially fraudulent recovery attacks if high volumes of attempts are made: we leave this as future work.

During recovery, users exchange unauthenticated messages with their recovery nodes. This enables users to initiate contact without knowing anything besides their username and password. Recovery nodes must implement rate-limiting to prevent against online brute-force attacks: for example, rate-limiting recovery requests by IP. This rate-limiting does not depend on HSMs: if at most t recovery nodes fail to rate-limit attempts, any attackers sending forged requests will still need to wait for the slowest recovery node among the $t+1$ reconstruction nodes to return a result before the attacker can check their attempt. Note that if IP rate-limiting is applied, attackers may be able to use a botnet or rent large blocks of IPs in the cloud to circumvent the rate-limit. However, rate-limiting instead per user may lead to denial-of-service attacks if attackers impersonate a user to use up their quota before they can submit a genuine recovery request. We leave this tradeoff as an area for future exploration.

Users are free to choose their reconstruction threshold t and the total number of recovery nodes n. In our prototype implementation's UI, we defaulted to $t = 3, n = 5$. Choosing $n \gg t$ is convenient for increased availability if some nodes are offline, but an adversary has more recovery nodes to choose from to potentially compromise and reconstruct s.

3.3 Dynamic Proactive Secret Sharing

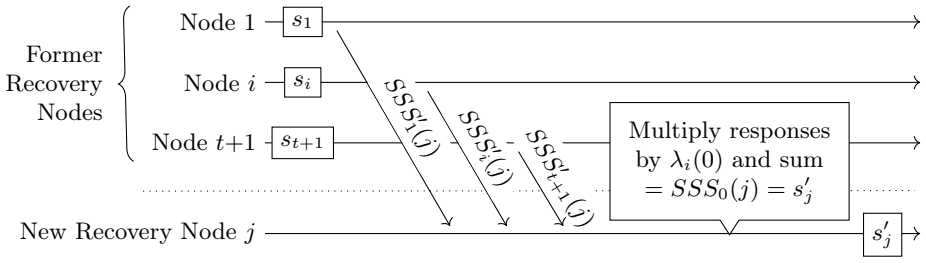
Dynamic proactive secret sharing (DPSS) combines two of the required properties: users must be able to update their set of recovery nodes, and former nodes must not be able to participate in recovering the user's key. DPSS is used to refresh each recovery node's secret share s_j used in the threshold OPRF.

We use the high-threshold Honey Badger approach by Yurek et al. [29]. Honey Badger is an *asynchronous* DPSS protocol, which means that it remains operational in the face of unpredictable network delays: this fulfills our final requirement for our choice of DPSS.

The core idea of Honey Badger is for each node i to generate a new SSS polynomial $SSS'_i(x) = s_i + c'_1 x + c'_2 x^2 \dots c'_{t'} x^{t'}$ with the new polynomial's constant term being the node's old secret share s_i. The node then broadcasts $SSS'_i(j)$ to each other node j, which allows node j to interpolate a new node share s'_j that represents the original nodes' shares at index j. These new node shares can be further interpolated to recover the original secret s. Recall that each $s_i = SSS(i)$ and $s = s_0 = SSS(0)$; likewise, each interpolated new share s'_i can be thought of as $SSS'_0(i)$, or alternative shares of $s_0 = s$. Thus, when these s'_i are interpolated

again, the original s is recovered. Intuitively, consider that the original secret s is split into shares once, with each of those node shares being split up again. This broadcast changes which nodes hold which sub-shares of the original secret, although the underlying shared data remains the same. $SSS'_i(x)$ can have a different degree, and therefore a different reconstruction threshold t', than $SSS(x)$, allowing users to add or remove recovery nodes. This secret refresh can also be configured to run at some desired interval (e.g. once per day) to protect against recovery nodes' shares being leaked over time.

Fig. 2. DPSS Secret Refreshing



Honey Badger is able to tolerate up to one-third of nodes being Byzantine and not following the protocol. The Kintsugi prototype currently does not implement these aspects of Honey Badger (e.g. the multi-valued validated Byzantine agreement), and hence requires all recovery nodes to correctly follow the protocol and only tolerates honest-but-curious or offline nodes. There is currently no way to determine which secret shares are valid for interpolation in order to recover s, which would be an issue in the presence of Byzantine recovery nodes. Kintsugi can be made Byzantine fault-tolerant in the future by implementing the remainder of the Byzantine agreement protocols.

Honey Badger requires a designated dealer node to calculate and broadcast the initial secret shares. In Kintsugi, the user whose key is being recovered acts as the dealer. This level of trust is acceptable, because the user's key is ultimately being recovered and we can assume the user behaves honestly. Applications that wish to avoid trusted setup can use a distributed key generation (DKG) protocol, such as the one proposed by Das et al. [12].

We also weighed other asynchronous DPSS protocols: in particular, we considered DyCAPS [13], a protocol with similar fault-tolerance guarantees and communication costs. We ultimately chose Honey Badger over DyCAPS due to Honey Badger's support of the Ristretto curve in existing OPRF instantiations and ease of development, whereas DyCAPS requires a pairing-friendly curve, such as the BLS12-381 curve, and would have required a custom OPRF design.

3.4 Changing Recovery Nodes

When users change their recovery nodes or their recovery threshold t, they initiate a DPSS refresh. Users then update their DHT entry of recovery nodes to hold the new set of recovery nodes, and the former recovery nodes destroy their old secret shares.

Former recovery nodes cannot participate in current key recovery attempts, thanks to the DPSS refresh protocol. Formerly valid secret shares, interpolated with current, refreshed shares, will fail to reconstruct s. For additional security, former recovery nodes are expected to cooperate and delete their old shares. Otherwise, it is possible for $t_{old}+1$ former recovery nodes to collude and recover s, for the threshold t_{old} at the point when the former recovery nodes were valid. Still, colluding recovery nodes cannot gain any information about P without mounting a brute-force attack.

4 Discussion

Kintsugi's approach provides several key benefits:

- **Decentralization.** Authentication mechanisms like OPAQUE [28], and most other E2EE platforms (see Table 1), rely on a single point-of-failure server for recovery. Distributing secret shares across multiple recovery nodes mitigates the concern that the server may mount attacks (e.g. brute-force) on the user's password or that the server secret is leaked.
- **Recovery from lost devices.** Because Kintsugi is based on a password, users can access their recovery key backups on a different device, in contrast to recovery schemes requiring hard copies of data. By design, Kintsugi does not provide any means of recovery if the user loses their password.
- **Brute-force and collusion resistance.** Kintsugi makes it relatively safe to use a lower-entropy password. If the backup encryption key were directly derived from the password without an OPRF exchange, offline brute-force attacks on the user's password would be feasible due to the low entropy of most passwords, whereas the OPRF output point used as an encryption key in Kintsugi has higher entropy. Kintsugi also requires the participation of recovery nodes to reconstruct the key, so attackers cannot directly perform an offline brute-force without compromising at least $t + 1$ secret shares. On the other hand, the rate-limiting performed by recovery nodes protects against online brute-force attacks.

A limitation of Kintsugi is that users must still keep track of their password. However, passwords remain a common experience that many users are comfortable with, as opposed to storing recovery codes or hard copies of recovery files. Kintsugi's usage of an OPRF output based on a low-entropy password is a sweet spot between short, easy-to-remember means of recovery like PINs [11] and high-entropy mechanisms such as Bitcoin seed phrases [21].

Kintsugi's concept of recovery nodes is flexible: it also allows the user's contacts' devices to serve as recovery nodes, similar to the social recovery methods used on platforms like PreVeil [22] or Apple iCloud [8]. One could consider an instantiation of Kintsugi leveraging email, SMS, or chat apps instead of our use of libp2p to conduct the OPRF evaluations. The user's contacts would act as a human-enforced "social rate limit" to prevent online brute-force attacks, since contacts are unlikely to respond to high volumes of requests.

5 Related Work

Table 1 compares the recovery mechanisms and properties of several E2EE services. Blessing et al. [10] have highlighted several concerns with existing E2EE platforms, including risks of total account lockout due to improper storage of recovery codes or files used by apps like WhatsApp and 1Password. As shown in the table, few platforms use decentralized recovery mechanisms, and those relying on social contacts for recovery are often vulnerable to social engineering.

5.1 Social Recovery and Trust Distribution

Several E2EE services rely on social recovery to provide some aspect of trust distribution or decentralization. For example, Apple iCloud allows users to designate a single recovery contact who can help the user regain access to their account [8,10]. The account owner is asked some verification questions as authentication, and iCloud has a waiting period in place to alert the account owner to potentially malicious recovery attempts. Nevertheless, a trusted recovery contact is often close enough to the user to guess answers to verification questions. In addition, the failsafe timeout relies on the user being online during that period to notice any notifications. This approach is also centralized, as Apple is a required intermediary in the recovery process.

1Password, an E2EE password manager, also supports social recovery via Recovery Groups assigned to teams of users [3]. Each Recovery Group member is able to help users recover access to their account. The whitepaper notes that this recovery process should require some out-of-band verification of requests and that the onus is on Recovery Group members to avoid social engineering. As well, because each Recovery Group member is able to unilaterally recover access, a single malicious recovery contact can gain access to the user's verification requests via email access and ultimately access the user's account. This is less secure than threshold-based designs, where recovery contacts must also compromise the other contacts' secrets to reconstruct the user's recovery key.

PreVeil is another E2EE platform, focusing on email and file collaboration. Among the E2EE providers listed in Table 1, it is the only service that supports decentralized recovery. PreVeil supports the notion of Approval Groups, a threshold social key recovery scheme, but their whitepaper does not explain how these groups protect against approvers colluding to recover the key [22]. PreVeil also supports Express Account Recovery, a feature that requires two shares to

Table 1. Comparison of E2EE platforms and their recovery mechanisms.

			Method					Properties		
	Service name	Data recovered	Password*	Recovery codes	PIN	Recovery files	Social contacts	Resistant to social engineering	HSM-based	Decentralized
Messaging, email	Signal	Key	-	-	✓[11]	-	-	N/A	✓[11]	-
	WhatsApp	Account	✓[26]	✓[27]	-	-	-	-	✓[26]	-
	PreVeil	Key	-	-	-	-	✓[22]	N/A	-	✓[22]
File storage	Apple iCloud	Account	-	✓[7]	-	-	✓[8]	-	✓[6]	-
	MEGA	Account	-	✓[20]	-	-	-	N/A	-	-
Password manager	1Password	Account	-	✓[4]	-	✓[4]	✓[4]	-	-	-
	LastPass	Account	✓[18]	✓[18]	-	-	-	N/A	-	-
Misc. (not E2EE)	SSS [23]	Key	-	-	-	-	✓[23]	-	-	✓
	Bitcoin	Key	-	✓[21]	-	-	-	-	**	✓
	OPAQUE	Key	✓[28]	-	-	-	-	N/A	-	-
Our paper	Kintsugi	Key	✓	-	-	-	✓	✓	-	✓

* In this table, passwords refer to user-chosen, memorable secrets, whereas recovery codes are automatically generated. PINs are also user-chosen, but shorter and low-entropy.

** Hardware crypto wallets act as consumer-grade HSMs.

recover the key, with the user storing one share and the other being stored on the PreVeil server [17]. However, this requires the user to safely store their share in an accessible location and relies on PreVeil as a single trusted party in recovering the user's key.

Note that none of the E2EE platforms listed in Table 1 that support social recovery contacts are resistant to social engineering attacks. An adversary may trick social contacts into providing their secret shares or initiating a recovery operation. Alternatively, social contacts may collude to gain access to the user's account. Some prior work on decentralized key recovery provides protection against individual curious nodes, but not gossiping, honest-but-curious nodes [5].

5.2 OPAQUE

We drew inspiration from the password-authenticated key exchange (PAKE) protocol OPAQUE [28] for Kintsugi. OPAQUE provides a way for users to authenticate themselves to a server without revealing their password to the server in plaintext. OPAQUE specifies methods for registration and login: in Kintsugi's design, we reframe login as key recovery while applying similar OPRF flows. Also, in OPAQUE, a single malicious server can perform an offline brute-force attack to guess the user's password; in Kintsugi, a threshold of recovery nodes must collude before offline brute-force is possible.

5.3 Threshold OPRFs

Several other threshold OPRF-based systems exist. For instance, Jarecki et al. propose an updatable, oblivious key management system for encrypted storage systems [15], based on their prior threshold OPRF work [14]. Their work describes a key management service with which clients perform an OPRF exchange to encrypt data and its decentralized, threshold OPRF-based variant. Although their system utilizes proactive secret sharing, the system does not specify how recovery nodes can be dynamically changed. It also requires the joint, distributed generation of new secrets and a distributed multiplication protocol, whereas Kintsugi avoids the need for any distributed key generation by relying on the user to deal secret shares. However, their system supports rotation of the OPRF secret s such that the encrypted data can be exclusively decrypted by new keys, while Kintsugi only rotates the secret shares and maintains the same shared s.

Juicebox is another decentralized key recovery protocol, based on PIN authentication and threshold OPRF exchanges [24]. Its design distributes trust across "realms", representing independent service providers. Juicebox does not allow updating these realms or the user's recovery threshold t after registration, and it requires at least some of these realms to be HSM-based in order to rate-limit PIN guesses. Interestingly, it also allows for HSM-based realms to be supplemented by software-backed providers, as long as a sufficient threshold of realms is reached in total. This provides a more cost-effective and scalable

approach, compared to other methods which are entirely reliant on HSMs, like WhatsApp's and Signal's recovery schemes.

6 Conclusion

Decentralized recovery mechanisms for E2EE services mitigate the risks of typical, centralized recovery flows. Relying on trusted hardware under a single provider's control can pose concerns for applications requiring metadata privacy or lacking financial resources. In this paper, we propose Kintsugi, a decentralized recovery protocol that distributes trust over multiple recovery nodes. Future work may include implementing Byzantine fault-tolerance and exploring alternative instantiations, such as a PKI-based recovery flow instead of relying on rate-limiting. In addition, we plan to integrate Kintsugi as an extension module for the Automerge CRDT library as part of a project to support authentication [1,30]. Overall, Kintsugi offers a new outlook on secure recovery protocols, eliminating the need for centralized infrastructure while allowing users to recover from device loss and maintaining strong security properties.

Acknowledgments. Emilie Ma conducted this work as a visiting researcher at the University of Cambridge.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Automerge CRDT, https://automerge.github.io/
2. Kademlia DHT (2023), https://docs.libp2p.io/concepts/discovery-routing/kaddht/
3. 1Password: 1Password Security Design (2024), https://1passwordstatic.com/files/security/1password-white-paper.pdf
4. 1Password: If you forgot your 1Password account password or you can't unlock the app (2024), https://support.1password.com/forgot-account-password/
5. Anderson, J., Stajano, F.: On Storing Private Keys in the Cloud, vol. 7061, p. 98–106. Springer Berlin Heidelberg, Berlin, Heidelberg (2014). https://doi.org/10.1007/978-3-662-45921-8_16, http://link.springer.com/10.1007/978-3-662-45921-8_16
6. Apple: Escrow security for iCloud Keychain (2024), https://support.apple.com/en-gb/guide/security/sec3e341e75d/web
7. Apple: Set up a recovery key for your Apple Account (2024), https://support.apple.com/en-gb/109345
8. Apple: Set up an account recovery contact (2024), https://support.apple.com/en-gb/102641
9. Bernstein, D.J., Hamburg, M., Krasnova, A., Lange, T.: Elligator: elliptic-curve points indistinguishable from uniform random strings. In: Proceedings of the 2013 ACM SIGSAC conference on Computer & Communications Security. p. 967–980.

CCS '13, Association for Computing Machinery, New York, NY, USA (Nov 2013). https://doi.org/10.1145/2508859.2516734, https://dl.acm.org/doi/10.1145/2508859.2516734

10. Blessing, J., Hugenroth, D., Anderson, R.J., Beresford, A.R.: SoK: Web Authentication in the Age of End-to-End Encryption (arXiv:2406.18226) (Jun 2024). https://doi.org/10.48550/arXiv.2406.18226, http://arxiv.org/abs/2406.18226, arXiv:2406.18226 [cs]
11. Connell, G., Fang, V., Schmidt, R., Dauterman, E., Popa, R.A.: Secret Key Recovery in a Global-Scale End-to-End Encryption System. In: 18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24). pp. 703–719. USENIX Association, Santa Clara, CA (Jul 2024), https://www.usenix.org/conference/osdi24/presentation/connell
12. Das, S., Xiang, Z., Kokoris-Kogias, L., Ren, L.: Practical asynchronous high-threshold distributed key generation and distributed polynomial sampling. In: 32nd USENIX Security Symposium (USENIX Security 23). pp. 5359–5376. USENIX Association, Anaheim, CA (Aug 2023), https://www.usenix.org/conference/usenixsecurity23/presentation/das
13. Hu, B., Zhang, Z., Chen, H., Zhou, Y., Jiang, H., Liu, J.: DyCAPS: Asynchronous Dynamic-committee Proactive Secret Sharing (2022/1169) (2022), https://eprint.iacr.org/2022/1169, publication info: Preprint.
14. Jarecki, S., Kiayias, A., Krawczyk, H., Xu, J.: TOPPSS: Cost-Minimal Password-Protected Secret Sharing Based on Threshold OPRF. In: Gollmann, D., Miyaji, A., Kikuchi, H. (eds.) Applied Cryptography and Network Security. pp. 39–58. Springer International Publishing, Cham (2017). https://doi.org/10.1007/978-3-319-61204-1_3
15. Jarecki, S., Krawczyk, H., Resch, J.: Updatable Oblivious Key Management for Storage Systems. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. p. 379–393. ACM, London United Kingdom (Nov 2019). https://doi.org/10.1145/3319535.3363196, https://dl.acm.org/doi/10.1145/3319535.3363196
16. Krassovsky, S., Cadden, G.: How WhatsApp is enabling end-to-end encrypted backups (Sep 2021), https://engineering.fb.com/2021/09/10/security/whatsapp-e2ee-backups/
17. Laroche, G.: (2024), https://www.preveil.com/blog/product-release-selective-sync-account-recovery/
18. LastPass: Recover your lost master password for LastPass (2024), https://support.lastpass.com/s/document-item?language=en_US&bundleId=lastpass&topicId=LastPass%2Frecover-master-password.html&_LANG=enus
19. Maymounkov, P., Mazières, D.: Kademlia: A Peer-to-Peer Information System Based on the XOR Metric. In: Revised Papers from the First International Workshop on Peer-to-Peer Systems. p. 53–65. IPTPS '01, Springer-Verlag, Berlin, Heidelberg (2002)
20. MEGA: What is my MEGA recovery key? (Dec 2021), https://help.mega.io/accounts/password-management/recovery-key
21. Palatinus, M., Rusnak, P., Voisine, A., Bowe, S.: Mnemonic code for generating deterministic keys (2013), https://github.com/bitcoin/bips/blob/master/bip-0039.mediawiki
22. PreVeil: PreVeil Security and Design (Jul 2023), https://www.preveil.com/wp-content/uploads/2019/10/PreVeil_Security_Whitepaper-v1.5.pdf

23. Shamir, A.: How to share a secret. Commun. ACM **22**(11), 612–613 (Nov 1979). https://doi.org/10.1145/359168.359176, https://dl.acm.org/doi/10.1145/359168.359176
24. Trapp, N.: Key to Simplicity: Squeezing the hassle out of encryption key recovery (Apr 2024), https://juicebox.xyz/blog/key-to-simplicity-squeezing-the-hassle-out-of-encryption-key-recovery
25. de Valence, H.: Ristretto - The Ristretto Group, https://ristretto.group/
26. WhatsApp: Can't remember password for encrypted backup (2022), https://faq.whatsapp.com/639067727894647
27. WhatsApp: How to reset your two-step verification PIN (2022), https://faq.whatsapp.com/2183055648554771/
28. Wood, C., Bourdrez, D., Lewi, K., Krawczyk, H.: The OPAQUE Augmented PAKE Protocol (2024), https://cfrg.github.io/draft-irtf-cfrg-opaque/draft-irtf-cfrg-opaque.html
29. Yurek, T., Xiang, Z., Xia, Y., Miller, A.: Long Live The Honey Badger: Robust Asynchronous DPSS and its Applications. In: 32nd USENIX Security Symposium (USENIX Security 23). pp. 5413–5430. USENIX Association, Anaheim, CA (Aug 2023), https://www.usenix.org/conference/usenixsecurity23/presentation/yurek
30. Zelenka, B., Good, A.: Beehive lab notebook (2024), https://www.inkandswitch.com/beehive/notebook/

William Wayman: When the user is setting out the nodes that they want to share the secrets with, is there a central trusted party that has knowledge of these nodes?

Reply: The user has knowledge of the nodes, and this is then replicated in a distributed hash table.

William Wayman: So the user needs to remember these nodes?

Reply: No, this DHT is replicated across all of the Kintsugi nodes. The user can access these nodes later by querying any recovery node.

William Wayman: So there's no central party that knows the nodes that a given user wants to access. There's no representation of that.

Reply: Yes, there's no central authority because this list is distributed across all other nodes. The distributed hash table holds these mappings.[1]

Fabio Massacci: I like all your examples, but what is the human's interface in your protocol? All of your first examples were a human that has lost their device. Here, you're instead making an example where the user needs a computation device that can compute an OPRF, so you need another device to actually recover the keys and do all these other things.

Reply: Recovery takes place locally on the user's device.

Fabio Massacci: So you need a second device to actually recover the first device that has been lost.

Reply: Yes, there needs to be recovery servers or other end user devices registered as recovery nodes. We assume the user has just bought a new device that they want to recover their data onto and that the recovery flow and its OPRF computation will take place on this new device.

Jonathan Anderson: If you're not requiring the recovery nodes to do authentication, then if those nodes are being used a lot, how do you do the rate-limiting? If there's 10,000 users who are using recovery node 3, what kind of rate-limiting does it do?

Reply: The rate-limiting is done based on the purported user that's attempting recovery, which is limited locally on that recovery node to once every X seconds or so.

Jonathan Anderson: That means an attacker could perform a denial of service attack against legitimate recovery attempts by pretending to be me, invoking the rate-limiting, but they wouldn't be able to make lots of password guesses. Is that right?

Reply: Yes, I believe so.[2]

---

[1] We updated the paper to additionally clarify that there is currently no confidentiality nor integrity guarantee for recovery node information.

[2] We are actively brainstorming how to more robustly perform rate-limiting to also protect against DoS, and the paper proposes IP-based rate-limiting as an alternative

Fabio Massacci: So, as a recovery node, you need to know the identity of the user, because otherwise you'll never be able to put a rate limit on them.

Reply: You just need to know the purported identity. User Y could be attempting to recover User X, for example.

Mark Lomas: So if I've understood this correctly, the user knows that they've succeeded in running this protocol, because they've got their data back. But does anyone other than the user know that it succeeded?

Reply: No.

Mark Lomas: Doesn't that worry you? Wouldn't it be useful to have an audit trail that says "Somebody appears to be trying to break into Mark Lomas' account"? We don't know whether it was actually Mark's device, or Susan breaking in.

Reply: I would imagine it would be easy to have an audit log that tracks recovery attempts made by purported users.

Mark Lomas: But you won't know the difference between a successful and a failed attempt.

Reply: But as a user, you'd be able to see, oh, this wasn't me.

Mark Lomas: The user could know, but none of the other components in the system would know.

Martin Kleppmann: (Co-author of paper.) If you wanted the recovery nodes to be able to find out whether a recovery attempt was successful, you could always have the user, after they decrypted the envelope, then send back a signed statement to the recovery notes to let them know after the fact that it was successful. And if no such statement arrives, then you assume that it was unsuccessful.

Fabio Massacci: I think the problem that Mark wants to solve is how someone else, not the user, can know if somebody's trying to forge the user's password. You can't assume the user is cooperating if no one's aware.

Tyler Moore: What would the advantages or disadvantages of adopting the system be, from the perspective of the platforms themselves? I can see how this has better security properties than existing efforts. That's good. But is there any advantage or disadvantage to say, Signal, adopting this versus their current protocol? It sounds like these recovery nodes could operate this decentralized infrastructure, or are you envisioning that the recovery nodes would be operated by the users themselves? What are the kind of the costs of operating this, compared to the current approaches? I'm just wondering if that could be an impediment to these services wanting to adopt your proposal.

Reply: The existing services already have a different type of decentralized infrastructure based on having all these HSMs, even if they're run on different

that comes with its own tradeoffs. In the meantime, however, we argue that genuine recovery attempts occur so infrequently that this is not a major concern.

cloud providers, under one company's control, as Signal does. We have to use decentralized recovery nodes, so you might have to spin up extra servers. As for the cost of running each of these recovery nodes, you can either use end user devices, so it could be that one Kintsugi user is acting as a recovery node for another Kintsugi user, or users can run their own recovery servers, sort of like Tor's hobbyist node operator model. These servers can all be operated by different service providers, which is a main advantage of the whole decentralization idea.

Liqun Chen: Is it true that under the secret sharing scheme, anyone else can request recovery but cannot get the right message back because they're not the original user? Does this mean the secret that the user shares cannot be a low-entropy password? Should it be a high entropy secret?

Reply: Due to the threshold OPRF construction, it doesn't matter what the user provides as their password, which becomes the OPRF secret. It's okay to use a lower entropy password because under the hood, the OPRF multiplies the secret by a random scalar, which provides more entropy. Then, assuming the discrete log problem, this becomes safe to combine with the recovery nodes' secret, so entropy is less of a problem.

Frank Stajano: At some point, you said that this method was impervious to social engineering. I am constitutionally skeptical about this type of claim. Could you please elaborate?

Reply: I said this in the context of existing social recovery schemes, where you tell your friend, "Hi, I lost my phone. Can you help get me back in?" Your friend has to decide if this is the genuine person that they know. With our scheme, because the recovery nodes operate blind to whatever the user has blinded their password to be, the recovery nodes don't have to make this judgment call on whether or not a recovery request was authentic.

Frank Stajano: On what basis does a recovery node make its decision to cooperate and apply its share?

Reply: It will always cooperate.[3]

Martin Kleppmann: The one thing that you have to trust the recovery nodes to do is to perform the rate limiting, so to not allow more than, say, one password guess per second per username.

Frank Stajano: So the claim could be rephrased as saying "It is totally socially engineerable, in the sense that they will always respond, but because there's a limit, they can't go very far."

Martin Kleppmann: Yes, and with every communication with the recovery nodes, the adversary can only try one password guess, whereas with a social

[3] We assume no Byzantine nodes. Byzantine fault-tolerance can be integrated into this protocol as a result of other protocol building blocks, but has not yet been implemented.

recovery system, if you manage to convince the recovery contact once, then you're in.

Frank Stajano: So "impervious to social engineering" could have meant "It is not possible to persuade the recovery nodes to cooperate. Instead they cooperate all the time, but we have a rate limit, and therefore it does not matter."

Tyler Moore: I think it means that her protocol does not rely on a human making a critical decision, which is the component which is inherently socially engineerable.

Tyler Moore: Last question: why do you call it Kintsugi?

Reply: Kintsugi is the Japanese art of mending together broken pottery with gold. One would fill in the cracks in a broken vase, for example, with gold in order to highlight the imperfections while making the vase better and new again. In this case, we're mending together our distributed key shares into this stronger, decentralized key recovery protocol. Thank you.