A Comparative Evaluation of End-to-End-Encrypted Key Retrieval in Backup Protocols

Dennis Funke
University of Wuppertal
Wuppertal, Germany
funke@uni-wuppertal.de

Kai Gellert
University of Wuppertal
Wuppertal, Germany
gellert@uni-wuppertal.de

Abstract

End-to-end encryption (E2EE) provides strong confidentiality guarantees to users by preventing service providers from accessing their data. At the same time, it introduces new operational challenges, most notably the restoration of an E2EE-protected backup on a new device after loss of the original device. In recent years, major instant messengers have deployed increasingly sophisticated key-retrieval schemes for encrypted backups, ranging from simple recovery codes to designs that depend on trusted hardware or advanced cryptographic primitives such as oblivious pseudorandom functions (OPRFs), threshold OPRFs, and secret sharing. While recent work has largely emphasized stronger cryptographic guarantees, the practical costs and trust assumptions implied by hardware dependencies and complex primitives are often not made explicit.

We present a structured framework for the comparative evaluation of E2EE backup key-retrieval schemes across usability, deployability, and security properties, and apply it to simple recovery codes, real-world deployments, and recent proposals from the literature, explicitly distinguishing recovery that depends on user-chosen low-entropy secrets from recovery using high-entropy material. Our evaluation shows that schemes whose recovery ultimately relies on user-chosen low-entropy secrets can provide substantially weaker resistance against large-scale adversaries than an approach using simple recovery codes. Even when assuming high-entropy recovery secrets, many recovery protocol designs introduce additional trust assumptions and potential single points of failure (e.g., compromise of specialized components or trusted setups) that can amplify the impact of large-scale adversaries. Overall, our results highlight that added system complexity often trades off against limited security gains under realistic threat models, motivating clearer accounting of cost and distribution of trust in future designs.

Keywords

end-to-end-encryption, encrypted backups, trusted hardware

1 Introduction

End-to-end encryption (E2EE) provides strong security guarantees by ensuring that service providers cannot access users' plaintext data. Yet, once E2EE is extended beyond messaging to long-term storage, new problems emerge. In particular, users need a way to restore an E2EE-protected backup on a new device when the original device is lost, stolen, or irreparably damaged.

We focus on *rare retrieval*. Backup recovery is intended to be used infrequently, yet it must work for non-expert users. This differs from authentication fallback flows that may occur more regularly and therefore optimize for different usability and security trade-offs. For example, it may be acceptable for backup recovery to require more effort (e.g., entering a long recovery secret or tolerating strict lockouts) because it is rarely invoked, whereas the same effort would be unacceptable for an authentication fallback. In contrast, mechanisms that are tolerable for frequent use (e.g., short secrets) may be too weak for rare retrieval unless additional safeguards are introduced. Accordingly, the properties that matter for E2EE backup recovery are not identical to those for fallback authentication [7, 17].

Rare retrieval also comes with a distinctive threat model. Backups are typically stored on infrastructure that may be compromised or malicious, and recovery mechanisms must remain secure even when large-scale adversaries with powerful infiltration capabilities can interact with them. At the same time, the user is part of the security. If recovery depends on user-chosen secrets, then real-world phenomena such as weak choices, misunderstanding of recovery prompts, or reuse across services can dominate the effective security of a scheme.

In recent years, major messaging platforms have deployed increasingly sophisticated key-retrieval protocols to enable password-based recovery of E2EE-protected backups. The WhatsApp E2EE backup protocol (WBP) combines an OPRF-based construction with rate limiting enforced by hardware security modules (HSMs) [12, 32]. Signal's Secure Value Recovery (SVR) distributes trust across multiple cloud trusted execution environments (TEEs) and uses secret sharing to raise the bar for compromise [11]. Threema Safe (TS) uses a password-based design with server-side operational rate limiting [29]. Beyond deployed systems, alternative designs have been proposed under varying trust assumptions, including the HSM-based refinements by Faller *et al.* [15], as well as protocols such as Juicebox [31] and Kintsugi [22].

A striking trend across both deployments and proposals is the push toward supporting low-entropy, user-chosen secrets (PINs or short passwords) by introducing stronger cryptographic machinery and/or trusted hardware to enforce guess limits [11, 12, 15, 22, 29, 32]. These techniques can indeed improve security against certain attackers, especially if online guessing can be reliably throttled. At the same time, they can introduce new threats: additional trust dependencies, operational complexity, and high-value components whose compromise may have system-wide consequences. Moreover, some practical risks are often out of model (e.g., user confusion about recovery prompts, or reuse-driven attacks such as credential stuffing), yet they can be severe in real deployments.

As security researchers, we carry a responsibility to make attacks as difficult as possible on all levels, not only in a security proof, but also under realistic large-scale threats. In particular, we should

Dennis Funke was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – project GE3741/1-1.

be cautious about introducing single points of failure that enable surveillance or compromise of many users at once. Centralized hardware-backed infrastructures and opaque setup procedures can become precisely such lucrative targets for large-scale adversaries: if compromised or subverted, they may enable attacks at a scale that is otherwise hard to achieve. Threshold designs and distribution of trust can mitigate this risk, but only if the trust domains are genuinely independent. This motivates the central question of this work: Are we improving user security in the rare-retrieval setting, or are we trading one class of risks (online guessing) for another (high-value centralized targets and opaque trust assumptions)?

To reason about these trade-offs systematically, we present a structured framework for evaluating E2EE key-retrieval schemes for rare retrieval along usability, deployability, and security dimensions. We apply this framework to (i) traditional recovery codes where the device generates a random code for the user to keep safe, (ii) real-world deployments in popular messengers, and (iii) representative proposals from the literature.

Our evaluation highlights a recurring pattern. When users choose low-entropy passwords, many sophisticated schemes hinge on additional trust assumptions (e.g., honestly executed trusted setups, confidence in the security of specialized hardware components). These assumptions can create high-value targets and new single points of failure, sometimes yielding real-world security that is not clearly better than traditional recovery codes, and in some cases worse under powerful adversaries. Even when users choose high-entropy secrets, sophisticated schemes may still introduce additional attack surface or trust dependencies compared to traditional recovery codes. This suggests that progress in this area of research should not be measured solely by adding stronger cryptographic properties in isolation, but by understanding how design choices change the overall security in practice, especially against large-scale attackers.

In summary, this work makes the following contributions:

- We develop a benefit-based evaluation framework for E2EE backup key-retrieval protocols that captures usability, deployability, and security properties tailored to rare retrieval.
- We provide simplified and unified protocol-core descriptions of traditional recovery codes, deployed key-retrieval protocols in instant messaging (including WBP, SVR, and TS), and prominent proposals from the literature (including the protocols by Faller *et al.* , Juicebox, and Kintsugi). We evaluate all schemes based on our benefit-based framework.
- We identify and discuss trade-offs, with particular emphasis on the security impact of trusted hardware, trusted setup, and other single points of failure that are especially attractive to large-scale adversaries.

Outline. The remainder of the paper proceeds as follows. In Section 2, we introduce our evaluation framework. In Section 3, we briefly recap the cryptographic building blocks used by the schemes we study. Section 4 alternates between simplified protocol-core descriptions and their evaluation under our framework. We discuss the resulting trade-offs and broader implications in Section 5, and conclude in Section 6.

Related Work. Given the breadth of work on authentication and account recovery, we scope this section to research that directly informs our setting and methodology. For broader background on authentication and recovery in the presence of E2EE, we refer the reader to [7].

Balsa *et al.* argue that deployed privacy technologies often do not eliminate trust in service providers, but instead rely on a broader chain of trust across the modern computing ecosystem [5]. Studying deployments such as encrypted messaging, they highlight that these technologies can shift trust toward a wider community of security and privacy experts who uncover and communicate failures, which in turn shapes the trust relationship between users and providers. This aligns with our focus on backup recovery, where additional infrastructure and operational assumptions become part of the trusted computing base.

Bonneau *et al.* systematically evaluate two decades of password-replacement proposals for web authentication using a set of usability, deployability, and security benefits [8]. They find that no known scheme comes close to achieving all desired benefits, and that many proposals fail because they overlook real-world constraints. Their benefit-based methodology serves as a benchmark. We adapt this style of evaluation to rare-retrieval schemes for E2EE backup recovery.

Blessing *et al.* study authentication and recovery challenges introduced by E2EE, where providers cannot assist users after forgotten secrets or device loss [7]. They highlight the resulting diversity of recovery mechanisms and argue that E2EE-based recovery becomes relevant even for mainstream services, since passkeys are device-bound and therefore require syncing and recovery across devices. Our work is not a broad survey of authentication and recovery mechanisms across the ecosystem. Instead, we zoom in on rare retrieval for E2EE backups in instant messaging and provide a structured comparison of selected deployed protocols and representative proposals, offering a complementary perspective centered on usability-deployability-security trade-offs in this setting.

2 Benefit-based Evaluation Framework

This section introduces a set of benefits for evaluating rare-retrieval protocols for E2EE backups in instant messaging. Our benefits are inspired by the usability/deployability/security framework of Bonneau *et al.* [8], but adapted to the backup-recovery setting. In particular, rare retrieval differs from everyday authentication: recovery is expected to be used infrequently, often under stress, and typically after device loss. Our benefit selection process was guided by the usability, deployability, and security objectives identified across all examined E2EE IM backup schemes, with the aim of establishing a common and comprehensive set of desirable properties. The rating criteria and their definitions were iteratively refined over the evaluation of the schemes to minimize overlap within each evaluation dimension and to cover salient features and distinctions among IM backup approaches.

We believe that the list of benefits and their precise definitions provides a robust framework for comparative evaluation of protocols in the IM backup context. We demonstrate and test this with the comparison provided in Section 4. Evaluation details and results are discussed in Section 5 that support the claim that the benefits

effectively highlight the strengths and weaknesses of the examined schemes. Note however that a list as such can always be expanded. For instance, the emergence of new requirements or security goals in the IM backup domain can be motivation to revise the evaluation benefits for future comparative work. Our selection process aligns with prior systematization efforts, including the seminal study by Bonneau *et al.* [8].

Interpreting Benefits. Each benefit captures a distinct desirable property. Importantly, the benefits are *not* intended to be naïvely summed into a single score. Different applications and threat models may prioritize different properties, and many benefits trade off against each other (e.g., avoiding memorization versus avoiding physical recovery artifacts). We therefore present benefits as a comparison mechanism rather than a ranking mechanism. For benefits where we observe fine-grained differences, we use a three-level scale (full/half/none). Otherwise, benefits are binary.

Pre-Authentication as a Modular Component. Several schemes rely on a pre-authentication step to prevent malicious clients from exhausting recovery attempts of honest users. Since pre-authentication mechanisms are largely orthogonal to the core of key retrieval, we treat them as an external factor that needs to be studied independently. Concretely, our goal is to compare the cryptographic and protocol-level key-retrieval mechanisms in isolation. We therefore model pre-authentication as an idealized black-box primitive that (when required) establishes that the requester is authorized to attempt recovery for a given account. In the evaluation, we assume access to an idealized pre-authentication mechanism that authenticates the client to the provider. The benefit *No-Pre-Authentication* captures whether such a module is required at all.

2.1 Usability-based Benefits

Rare-retrieval security is only meaningful if users can successfully recover. In practice, recovery failures are often caused by (i) forgetting secrets, or (ii) losing access to physical recovery material. We therefore include two usability dimensions that repeatedly dominate rare-retrieval outcomes: cognitive and physical burden.

Memorywise-Effortless. This benefit captures whether recovery requires the user to reliably remember a secret over long time spans. We grant the full benefit if the scheme does not require the user to memorize any secret [8]. We grant half the benefit if the user only needs to remember a short secret, such as a PIN. A scheme does not receive this benefit if it requires memorizing a long secret.

Physically-Effortless. This benefit captures whether recovery requires managing physical artifacts (e.g., writing down, printing, storing, and later retrieving recovery material). We grant the benefit if backup and recovery require no physical effort beyond simple interactions (e.g., pressing a button). Schemes that require managing recovery material do not receive the benefit.

Free-Provider-Choice. A scheme grants this benefit if it can be deployed with multiple independent operators without changing the protocol and without requiring a single global trust anchor (e.g., a single vendor, single service operator, or single infrastructure root). A schemes receives half the benefit if it allows the selection of centralized backup servers on a granular level. We do not grant the benefit if the user has no choice regarding the storage of key material.

2.2 Deployability-based Benefits

Deployability is central for rare-retrieval mechanisms because recovery must work when needed, across large populations and over long time periods. In addition, deployability choices often determine the trust anchors of a system (e.g., dependence on specialized infrastructure).

No-Special-Hardware-Required. We grant this benefit to schemes that do not require specialized hardware (e.g., HSMs or TEEs) and can be deployed using commodity hardware on both the client and server side. Schemes that rely on specialized hardware do not receive this benefit.

Deployed-at-Scale. We grant this benefit if the scheme has been implemented and deployed at large scale for real backup use beyond research prototypes. Schemes for which we have not (yet) observed practical deployments do not receive this benefit.

Non-Proprietary. We grant this benefit if the scheme can be implemented and used for any purpose without royalties or reliance on proprietary techniques. In particular, the relevant methods should be publicly documented. Open-source implementations are a strong indicator but not strictly required.

2.3 Security-based Benefits

Our security benefits reflect the threats that dominate rare-retrieval for E2EE backups: guessing attacks against low-entropy secrets, theft of recovery material, trust dependencies, and compromise of stored recovery state. We additionally include benefits capturing post-quantum migration, trusted setup, and the existence of a formal security analysis.

Resilient-to-Throttled-Guessing. In our setting, this benefit captures whether online guessing remains infeasible when the adversary can interact with the recovery interface but is effectively rate-limited (e.g., by per-account counters). We expect this to be solved on a protocol level and not via external measures. We require protocol-level rate limiting because operational measures are difficult to specify and evaluate, and they become brittle under compromise or adversarial control of infrastructure, precisely the setting rare retrieval aims to handle. Thus, we only grant *Resilient-to-Throttled-Guessing* when rate limiting is an explicit protocol guarantee (possibly enforced by trusted hardware).

Resilient-to-Unthrottled-Guessing. We grant this benefit if an attacker whose guessing rate is limited only by available computing resources cannot successfully guess the secrets of a significant fraction of users. Schemes do not receive this benefit if the credential space is too small to withstand brute-force search, including dictionary attacks, rainbow tables, and other optimized guessing methods that apply when credentials are user-chosen.

Resilient-to-Theft. In the backup-encryption setting, we grant this benefit to schemes in which stealing a physical object (e.g., a printed recovery code or storage device) does not enable the thief to recover the user's chat history in plaintext.

No-Trusted-Third-Party. We grant the full benefit if the scheme does not require trusting any third party to keep the user's backup

secure. We grant half the benefit if trust is distributed across multiple third parties. Schemes do not receive this benefit if the user must trust one third party to keep the backup secure.

Post-Quantum-Secure. We grant the full benefit if the scheme is post-quantum secure out of the box or generically built from components for which quantum-safe alternatives exist. Schemes do not receive this benefit if they cannot be easily upgraded to resist adversaries with a quantum computer.

No-Pre-Authentication. We grant this benefit to schemes that do not require any pre-authentication step. Schemes that depend on a pre-authentication protocol to function correctly do not receive this benefit.

No-Trusted-Setup. We grant this benefit to schemes that do not rely on a trusted setup to keep backups secure. Schemes that can leak secret values if the trusted setup was compromised, do not receive this benefit.

Security-Proof . Formal security analysis increases confidence and clarifies assumptions. We grant the benefit if a formal security proof exists. Schemes without one do not receive this benefit.

Resilient-to-Storage-Corruption. This benefit captures the impact of server or HSM storage corruption on an adversary's ability to attack the scheme. We assume that, upon storage corruption, all client-specific data stored by the server or HSM is leaked to the adversary in a read-only manner. We further assume that storage corruption occurs between backup registration and recovery. A scheme receives the full benefit if it remains secure under storage corruption. It receives half the benefit if, after corruption, an adversary must still perform an additional offline guessing attack to recover the key or backup. Schemes do not receive this benefit if storage corruption renders them insecure.

3 Building Blocks

In this section, we briefly recap the building blocks that are used in several key retrieval protocols.

Symmetric Encryption. A symmetric encryption scheme consists of two algorithms SEnc and SDec. Algorithm SEnc takes a symmetric key $k_{\mathsf{sym}} \in \{0,1\}^\lambda$ and a message m as input, and outputs a ciphertext c (using internal randomness and, if applicable, a nonce). Algorithm SDec takes a symmetric key $k_{\mathsf{sym}} \in \{0,1\}^\lambda$ and a ciphertext c as input, and outputs either the message m or an error symbol $\perp$ if decryption fails. Unless stated otherwise, we assume SEnc/SDec provides authenticated encryption, i.e., confidentiality and ciphertext integrity.

Public-Key Encryption. A public-key encryption scheme consists of three algorithms PKEGen, Enc, and Dec. Algorithm PKEGen generates a key pair (pk, sk) consisting of a public key pk and a secret key sk. Algorithm Enc takes a public key pk and a message m as input, and outputs a ciphertext c (using internal randomness). Algorithm Dec takes a secret key sk and a ciphertext c as input, and outputs either the message m or an error symbol $\perp$ if decryption fails. Unless stated otherwise, we assume the scheme is IND-CCA secure.

Digital Signatures. A digital signature scheme consists of three algorithms SigGen, Sign, and Verify. Algorithm SigGen generates a key pair $(pk_{\mathsf{sig}}, sk_{\mathsf{sig}})$ consisting of a public verification key pk_{sig} and a secret signing key sk_{sig}. Algorithm Sign takes the secret key sk_{sig} and a message m as input, and outputs a signature σ. Algorithm Verify takes the public key pk_{sig}, a message m, and a signature σ as input, and outputs 1 if the signature is valid for message m under pk_{sig} and 0 otherwise. Unless stated otherwise, we assume the scheme is existentially unforgeable under chosen-message attacks (EUF-CMA).

Hash Functions. A hash function is an efficiently computable algorithm H that maps inputs of arbitrary length to fixed-length outputs, i.e., $H : \{0,1\}^* \to \{0,1\}^\lambda$. We typically require H to be collision resistant, meaning it is computationally infeasible to find distinct inputs $x \neq y$ such that $H(x) = H(y)$, and preimage resistant, meaning that given an output $h \in \{0,1\}^\lambda$ it is computationally infeasible to find an input x with $H(x) = h$.

Pseudorandom Functions. A pseudorandom function (PRF) is a keyed efficiently computable function PRF where $\mathsf{PRF} : \{0,1\}^\lambda \times \{0,1\}^* \to \{0,1\}^\lambda$. Informally, for a uniformly random key k, oracle access to $F(k,\cdot)$ is computationally indistinguishable from oracle access to a truly random function with the same domain and range. A PRF can be used to deterministically derive pseudorandom values from inputs while keeping the key k secret.

Secret Sharing. A (t,n)-threshold secret sharing scheme consists of two algorithms Share, Reconstruct. Algorithm Share takes a secret s and outputs n shares $s_1, \ldots, s_n$. Algorithm Reconstruct takes any set of at least $t+1$ shares and outputs the secret s. The scheme guarantees correctness for any $t+1$ valid shares. Any set of t or fewer shares reveals no information about s.

Oblivious PRFs. An oblivious PRF (OPRF) is a two-party protocol between a client and a server that lets the client obtain a PRF output $F(k,x)$ of an input x of its choice, where the server holds the secret key k. The client learns nothing else about k (beyond what is implied by the output), while the server learns neither x nor $F(k,x)$ (beyond what is implied by holding k). A threshold OPRF (T-OPRF) is a threshold variant of an OPRF where the server's key is shared among multiple parties [20] and a client is required to interact with a subset of them to evaluate the PRF.

Hardware Security Modules. A *hardware security module* (HSM) is a specialized tamper-resistant piece of hardware that can be programmed once and then locked to prevent subsequent modifications. This allows systems to "outsource" cryptographic key material and operations to the HSM. After initialization, the module performs cryptographic operations while ensuring that the key material never leaves the device. We note that an HSM usually signs all its messages, such that any interacting party in possession of the HSM's public verification key can verify the authenticity of all messages generated by the HSM.

Relying on an HSM shifts trust from the HSM operator to the security guarantees of the module itself, i.e., the HSM acts as trusted third party within the system. This can be advantageous since compromising the operator does not compromise the key material stored within the HSM. However, these guarantees only hold if (i) the programming and locking procedures (the so-called *trusted*

setup ceremony) of the HSM have been executed honestly, and (ii) the HSM indeed remains uncompromised. Observe that both conditions constitute single-point-of-failures; a compromised setup can leak key material, and an attack on the HSM product line could undermine many deployments simultaneously. This makes such trusted components attractive targets for large-scale adversaries.

Trusted Execution Environment. A trusted execution environment (TEE) is similar to an HSM, but instead of being a dedicated external device, it is typically integrated into modern CPUs. TEEs aim to provide HSM-like security properties at lower cost and broad availability by offering mechanisms such as verifiable launch, run-time isolation, trusted IO, and secure storage [27].

A particularly relevant feature for key retrieval protocols is remote attestation. It was introduced to mitigate scenarios in which an attacker can remotely install or modify software on a general-purpose system that contributes to security-critical interactions between independent parties [10]. Remote attestation allows a device to present cryptographic evidence about the software state of a protected execution context (e.g., an enclave), enabling a remote party to decide whether to proceed based on expected measurements. Attestation is not limited to one direction. In a client-server setting, both client and server can require such evidence before proceeding.

4 Evaluation of Rare Retrieval Protocols

In this section, we examine which benefits are achieved by existing rare-retrieval protocols. This includes recovery codes, real-world deployments such as WhatsApp's end-to-end-encrypted backup protocol [12, 32], Signal's Secure Value Recovery [11], and Threema Safe [29], as well as protocols proposed in the literature, including the three designs by Faller *et al.* [15], Juicebox [31], and Kintsugi [22].

We exclude Telegram from our backup comparison because it does not offer E2EE backups for secret chats, and we treat Apple's iMessage backup ecosystem as out of scope because backup recovery is tightly coupled to iCloud backup and Apple-ID account recovery and Apple does not provide a protocol-level specification of the underlying key-retrieval mechanisms that would allow us to isolate and evaluate it as a standalone rare-retrieval scheme comparable to the protocols in this paper. In addition, we also exclude the Labyrinth protocol used by Facebook Messenger, as it relies on the WhatsApp backup protocol and recovery codes as building blocks [7, 24].

Each subsection follows the same structure. We first present a simplified description of the protocol's cryptographic core, and then discuss the benefits it achieves. The protocol descriptions are deliberately simplified by omitting variables and steps that are not relevant to the benefits we analyze. This includes modeling artifacts introduced by security models such as the universal composability framework [9], as well as auxiliary values used only to identify a specific client among many. We provide a detailed explanation of our simplifications in Section 4.8.

Depending on the user's choice to use a high-entropy recovery secret, or to use a low-entropy memorized password, the evaluation of some benefits may change. If the protocol allows a user-chosen low-entropy password, we first evaluate this mode of operation. At the end of this section, we explain how switching to a high-entropy recovery secret (either user-generated or application-generated) changes our evaluation. A detailed summary of our evaluation can be found in Table 1 in Section 5.

4.1 Recovery Codes

A simple and long-established approach to rare-retrieval protocols is to encrypt the backup under a device-generated, high-entropy symmetric key and have the user store that key externally, typically by printing it or writing it down, and keeping it in a safe place. We refer to this family of approaches as *recovery codes*.

We note that the user-held code in such approaches is referred to by various names, including "long-term recovery key", "recovery key", "recovery code", and "master passphrase" [7]. In this work, we use the term high-entropy recovery secret to denote this pseudorandom recovery material. Unlike a conventional user-chosen password, it is intended to provide sufficient entropy to resist brute-force guessing.

Recovery codes are well established and have been used in many applications for years. Blessing *et al.* find that high-entropy recovery codes are the primary backup method among web authentication schemes, yet they also observe little consensus on which rare-retrieval protocols to adopt in E2EE settings [7]. This lack of consensus suggests that the practical benefits and trade-offs of different rare-retrieval protocols are not yet well understood.

Beyond instant messaging, recovery codes also appear in storage and email services as well as credential managers. In the instant messaging ecosystem, prominent examples include WhatsApp,[1] which generates a 64-digit encryption key as a long string of random characters, Signal,[2] which provides a 30-digit passphrase for the user to store safely, and Element, which generates a 48-character recovery key for secure backup of message keys and related verification material[3].

Since recovery codes are prevalent across web authentication—albeit with subtle variations and under different names—we summarize what the protocols have in common and highlight best practices.

Key Generation. Since the user is responsible for storing the recovery code (and we assume it is kept in a safe place), the length of the recovery code is not constrained by memorization capabilities. This enables the use of a high-entropy secret (e.g., 256 bits) generated uniformly at random. Crucially, the recovery code should be generated on the user's device using a cryptographically secure random number generator, rather than being chosen by the user. User-chosen secrets are prone to credential-stuffing attacks, where leaked credentials are tried against other services [4]. While recovery codes are not traditional password logins, the underlying risk is analogous. Reuse or predictable choice can turn a single leak into a broader compromise. Sometimes user interaction (e.g., timing and touch events) is used as supplementary input to collect sufficient entropy for key generation.

[1] https://faq.whatsapp.com/490592613091019/
[2] https://support.signal.org/hc/en-us/articles/10066926526362-Android-On-Device-Backups
[3] https://element.io/en/help

Symmetric Encryption. Backup encryption and decryption are performed locally on the user's device, while the encrypted backup itself is typically stored in untrusted infrastructure (e.g., a cloud service) that an adversary may be able to access. In an E2EE setting, the goal is that only the user can obtain the backup in plaintext, even if the ciphertext is exposed. Consequently, backups should be protected with a well-studied symmetric encryption scheme and an appropriate key length (e.g., AES-256) to render brute-force attacks infeasible, including large-scale adversaries with access to quantum computers [6, 16, 25].

Backup Storage. The encrypted backup can be stored on any medium, including untrusted cloud storage. In the extreme, it could even be made public: without the recovery code, decryption should remain computationally infeasible. In practice, storage choices differ accross applications. Some, such as WhatsApp, tie backups to platform cloud services (e.g., Google Drive or iCloud) associated with a user account [12]. Others, such as Signal, export an encrypted backup file and leave managing the storage to the user. From a deployability and usability perspective, a reasonable best practice is to offer a convenient solution (e.g., cloud upload) while also supporting manual export to arbitrary storage media to improve flexibility and interoperability.

Key Management. When using recovery codes, the user is responsible for managing and protecting the recovery code. A crucial prerequisite is that the code can be stored securely for long periods of time. If the code is printed or written down, this requires safe physical storage—an assumption that may not hold for all users. For instance, high-risk users (e.g., investigative journalists, activists, or whistleblowers) may not have storage safe from illegitimate access. However, considering that the majority of users have other important paper documents (e.g., birth certificates or school diplomas) stored in some safe place, they should also be able to safely store a recovery code.

An alternative is to store the recovery code digitally, for example in an E2EE credential manager. This shifts trust from the user's physical security to the credential manager's security and availability. Recent work has found that many credential manager protocols are vulnerable to offline guessing attacks [13]. While using a high-entropy secret can mitigate guessing attacks, it also reintroduces the practical storage challenges discussed above. In fact, most credential managers support recovery codes for authentication or recovery [7]. Since the security of credential managers is an active research area and orthogonal to our focus, we assume physical storage of the recovery code for this work.

Rare Retrival. Managing recovery codes is cumbersome for users [7]. However, recovery mechanisms are intended to be used infrequently and can therefore justifiably require more effort than everyday authentication. In practice, recovery codes are mostly a fallback when more convenient options are unavailable or fail [7]. Many instant messaging applications provide device-to-device migration mechanisms, for example, sending an encrypted backup to a nearby device (e.g., via Bluetooth) and supplying the key material via a QR code scanned by the new device.

To support regular backups, an application may store the recovery code on the device itself, where messages are accessible in plaintext anyway, enabling automated backup creation. Depending on the user's storage choice, the encrypted backups can then be uploaded automatically. Only in the rare event that a device is lost, stolen, or irreparably damaged, the user must retrieve the recovery code to import the backup on a new device.

Usability Improvements. Developers have several options to improve usability when using recovery codes. Instead of displaying an alphanumeric string, the key can be encoded as a word-based passphrase drawn from a dictionary. Compared to random character strings, such phrases be entered faster and with fewer errors [28]. Furthermore, they are easier to transcribe by hand since users can recognize familiar words. In addition, speech-to-text software may further reduce entry effort and improve usability.

If users print the recovery material rather than writing it down, it can also be encoded as a machine-readable QR code. Most mobile devices can scan QR codes, which again reduces transcription errors. Some applications attempt to keep users familiar with the recovery process by prompting them to re-enter the recovery code periodically. For example, WhatsApp regularly asks users to enter the recovery code when accessing the app [7]; other instant messengers provide similar reminders as an optional feature. It is unclear how effective these reminders are, or if they even harm security.

We further remark that users might want to screenshot their displayed recovery code. This could lead to loss of the backup if the screenshot is not securely transmitted to another device. Furthermore, automatic synchronization could expose the recovery code if the screenshot is uploaded to a cloud storage in plaintext. A measure to protect the user from screenshotting is to disable screenshots of the recovery code entirely.

4.1.1 Evaluation. Table 1 in Section 5 summarizes the benefits achieved by recovery codes under our framework.

We assume that the client device generates a high-entropy recovery secret, which the user prints out (or writes down) and stores in a safe place. Hence, the scheme does not provide the *Physically-Effortless* benefit. In turn, since the user is not required to memorize any secret, it provides the *Memorywise-Effortless* benefit. The scheme provides the *Free-Provider-Choice* benefit because the client does not have to store key material with a predefined server.

Recovery codes rely only on local symmetric encryption and therefore require no specialized hardware, yielding *No-Special-Hardware-Required*. Since recovery-code backups have been deployed in practice for many years, the scheme provides *Deployed-at-Scale*. Moreover, recovery-code mechanisms are widely documented and implemented (including open-source implementations, e.g., by Signal), so we grant *Non-Proprietary*.

Because the recovery secret is device-generated and high entropy, guessing attacks are infeasible. Accordingly, both throttled and unthrottled guessing succeed only with negligible probability, and the scheme provides *Resilient-to-Throttled-Guessing* and *Resilient-to-Unthrottled-Guessing*. However, since the recovery secret is stored on a physical medium that can be stolen, the scheme lacks *Resilient-to-Theft*. No third party is required to keep the backup secure, so the scheme provides *No-Trusted-Third-Party*.

Regarding post-quantum considerations, recovery codes can be post-quantum secure out of the box by choosing an appropriate

symmetric security level (e.g., sufficiently long keys), and thus we grant full *Post-Quantum-Secure*. Since the scheme operates offline, it does not require any pre-authentication, and it does not depend on any trusted setup beyond the client device. Therefore it provides *No-Pre-Authentication* and *No-Trusted-Setup*. Recovery codes can be trivially proven secure, since no server-interaction is required. Thus the scheme provides the *Security-Proof* benefit. Finally, the scheme is resilient to storage corruption because no key material is stored on any server, and thus it provides *Resilient-to-Storage-Corruption*.

4.2 WhatsApp Backup Protocol

In order to enable E2EE for backups, WhatsApp deployed the Whatsapp Backup Protocol (WBP) in late 2021 [12, 32]. A key design goal of the WBP is to allow users to protect their backup with a low-entropy password. Since low-entropy passwords are vulnerable to dictionary and brute-force attacks when used directly for encryption, the WBP does not encrypt backups under the password itself. Instead, it employs a password-protected key retrieval (PPKR) scheme backed by HSMs, which enforce a strict limit on online password guesses.

Intuitively, the WBP makes a weak password "usable" by ensuring that an attacker cannot try arbitrarily many guesses during a recovery. This rate limiting is enforced inside the HSM. Davies *et al.* [12] provide a formal security analysis of the WBP and show that it achieves most of its intended security goals under their model. However, their analysis also identifies an edge case in which an attacker can increase the number of allowed guesses. It requires compromising the WhatsApp server that relays messages between client and the HSM and relies on the client initializing the WBP multiple times with the same password.

The protocol flow of the WBP is depicted in Figure 1. During initialization, the client interacts with the HSM (via the WhatsApp server as relaying intermediate) to jointly execute the OPRF. The client inputs a user-chosen password pw while the HSM uses a client-specific OPRF key k_{OPRF}. As a result, the client derives an export key k_{sym}, while the HSM stores an associated "envelope" value env that is needed for later recovery. The client then samples a random backup key k_{backup} and encrypts it under k_{sym}, obtaining a ciphertext e. The client sends e to the HSM, which stores it along with env, k_{OPRF}, and a counter ctr that tracks remaining recovery attempts (initialized to ten).

During recovery, the client again interacts with the HSM to perform the OPRF evaluation again, now on an entered candidate password pw'. Before proceeding, the HSM checks the attempt counter ctr, decrements it, and enforces lockout if no attempts remain (i.e., it deletes env, e). The OPRF evaluation outputs a value k'_{sym} to the client as well as values shk' to the client and shk to the HSM. The OPRF guarantees that if and only if the client provides the correct password $pw' = pw$, then $k'_{\mathsf{sym}} = k_{\mathsf{sym}}$ and $\mathsf{shk}' = \mathsf{shk}$. The client proves knowledge of the correct password by presenting shk'. If the HSM verifies equality, it resets the counter and releases the stored ciphertext e to the client. Finally, the client decrypts e using k'_{sym} and recovers the backup key k_{backup}.

4.2.1 Evaluation. Table 1 in Section 5 summarizes the benefits achieved by the WBP under our framework.

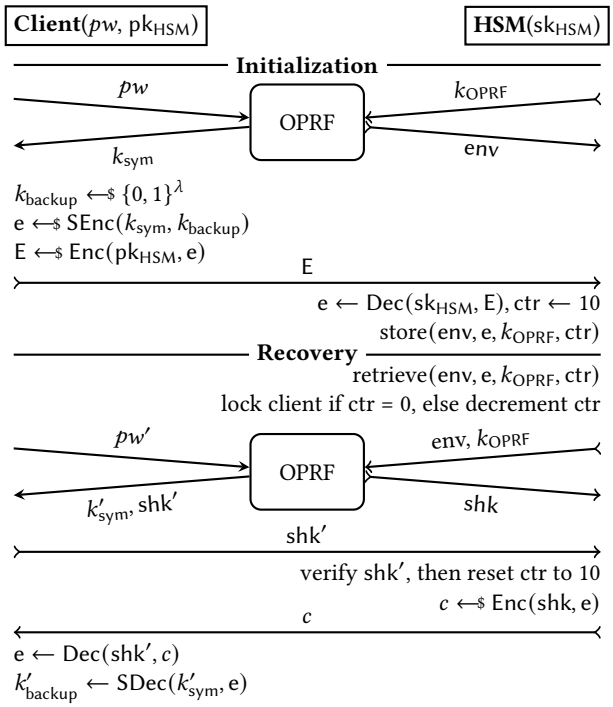


Figure 1: Simplified execution of the WBP. The arrows $\longrightarrow$ and $\longleftarrow$ indicate that the messages are forwarded by a WhatsApp server.

When the WBP is used with a memorizable low-entropy password, it provides the *Physically-Effortless* benefit but only half of the *Memorywise-Effortless* benefit, since the user must memorize a short secret. As the key material must be stored with the central HSM the scheme is not granted *Free-Provider-Choice*. Due to the reliance on HSMs it does not provide *No-Special-Hardware-Required*. It provides *Deployed-at-Scale*, as it has been deployed at large scale since late 2021. However, since WhatsApp is proprietary and closed source, we do not grant *Non-Proprietary*.

The WBP is resilient to throttled guessing because the HSM enforces a strict limit of ten password-guessing attempts. If this safeguard is bypassed or ineffective, an unthrottled adversary can brute-force the low-entropy password. Hence, the WBP provides *Resilient-to-Throttled-Guessing* but not *Resilient-to-Unthrottled-Guessing*. Since there is no physical recovery component whose theft would enable decryption, the WBP provides *Resilient-to-Theft*. At the same time, enforcing the guessing limit relies on a single third-party component (the HSM infrastructure), so the WBP does not provide *No-Trusted-Third-Party*.

The WBP uses an OPRF in a non-black-box way [12, 15]. While hybrid combiners exist to upgrade OPRFs to post-quantum security, they do not apply straightforwardly when the OPRF is not used as a black-box building block. We therefore do not grant *Post-Quantum-Secure*. Moreover, maintaining a per-user attempt counter requires the client to authenticate to the service before recovery, so WBP does not provide *No-Pre-Authentication*. Finally, WBP depends on trusted setup assumptions for the HSMs, and thus does not provide

No-Trusted-Setup. Since WBP has a formal security analysis [12], it provides *Security-Proof*.

Regarding storage corruption, WBP stores a client-specific state inside the HSM, including values such as env and k_{OPRF}. If this state is leaked, it can enable an offline password-guessing attack that verifies guesses via the OPRF-derived checks, which is feasible when users choose low-entropy passwords. We therefore grant only half of the *Resilient-to-Storage-Corruption* benefit.

4.3 Secure Value Recovery

Secure Value Recovery (SVR) is Signal's approach for enabling low-entropy, memorizable passwords to protect E2EE account backups [11]. At the time of writing, SVR is used to protect a backup of account information (e.g., user profile and social graph), but not message history or media. Similar to the WBP, SVR relies on secure hardware in the form of TEEs[4]. In its current third version, SVR is a multi-server key-recovery protocol that distributes trust across three TEE deployments backed by different hardware ecosystems and cloud providers: Intel SGX in the Microsoft Azure Cloud, AMD SEV-SNP in the Google Cloud, and Nitro in the AWS Cloud. The design is motivated by the observation that vulnerabilities in any single TEE are often patched quickly. The distribution of trust then makes a successful attack harder by requiring an attacker to exploit a vulnerability in all three TEEs within a time window simultaneously.

A simplified execution of SVR is depicted in Figure 2. During initialization, the client samples s uniformly at random from a field $\mathbb{F}$ and splits it into shares s_1, s_2, s_3 by using secret sharing. Each server i samples a random value $k_i \in \mathbb{Z}_m$ and runs the OPRF, where the client inputs its password pw. The OPRF outputs ρ_i are only learned by the client. The client then masks each share by computing $e_i = s_i \oplus \rho_i$, and derives $r||k_{\text{backup}} := H(0, s)$, where k_{backup} serves as the backup key and r as an auxiliary value. To enable verification during recovery, the client computes a checksum C that commits to the relevant initialization values. Finally, the client sets a limit of recovery attempts[5] and uploads the masked shares and recovery metadata $((e_1, e_2, e_3), C, \text{ctr})$ to each server.

During recovery, each server retrieves the client's record and enforces rate limiting. It checks whether attempts remain in ctr, decrements the counter, and locks the record (i.e., deletes client-specific state) once the counter is exhausted. Assuming recovery is allowed to proceed, the client runs the OPRF again with each server using a password guess pw', obtaining ρ'_i. If and only if $pw' = pw$, the client obtains matching masks $\rho'_i = \rho_i$. The servers then return the stored masked shares and checksum. Since the client receives responses from multiple servers, it can cross-check consistency (e.g., detect a faulty or malicious server that returns unexpected values). The client unmasks the shares via $s'_i = e_i \oplus \rho'_i$ and reconstructs s' from the resulting shares. Finally, the client verifies the checksum by recomputing it from the recovered values and comparing it to C. If the check passes, the client derived the correct key $k'_{\text{backup}} = k_{\text{backup}}$ and uses it to decrypt the backup.

[4]SVR gates access to the TEE-backed servers via SMS-based verification and uses remote attestation so the client can verify it is talking to the expected enclave code.
[5]Note that the client must choose this limit to be reasonable low for rate limiting to effectively protect low-entropy passwords, cf. https://signal.org/blog/secure-value-recovery/ recommending ctr = 5.

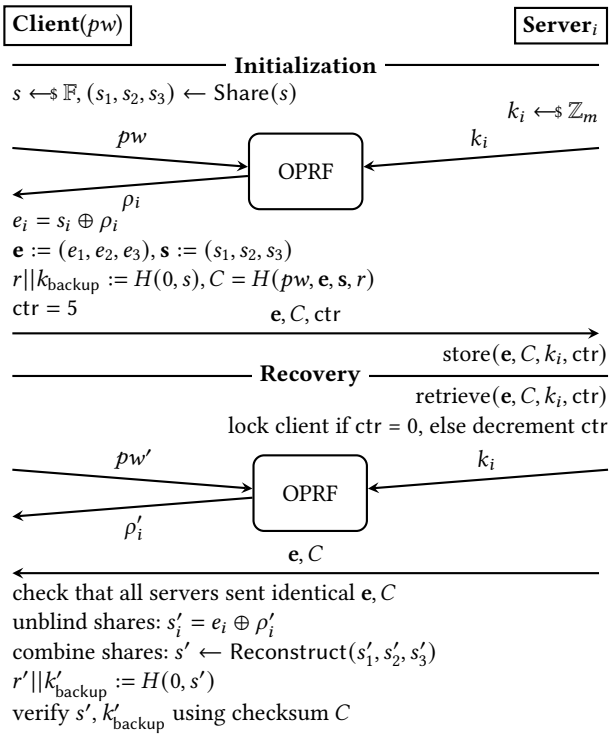


Figure 2: Simplified execution of the SVR protocol with $n = 3$ servers [11]. Channels are authenticated using TLS. *SSS* **denotes Shamir-Secret-Sharing and H is a hash-function.**

4.3.1 Evaluation. Table 1 in Section 5 summarizes the benefits achieved by SVR under our framework.

When used with a memorizable low-entropy password, SVR offers the same usability benefits as WBP. It provides *Physically-Effortless* since no physical recovery artifact needs to be managed, and only half of *Memorywise-Effortless* since the user must remember a short secret. Although SVR shares secrets across three TEEs, the user has no choice but to store the key material with all of them. Thus we do not grant the *Free-Provider-Choice* benefit. The TEEs are required to enforce a limit on password-guessing attempts. Hence, it does not provide *No-Special-Hardware-Required*. Since SVR has been deployed at scale for Signal users, we grant *Deployed-at-Scale*. Signal's implementation is available as open source under a free software license, so SVR provides *Non-Proprietary*.

SVR is resilient to throttled guessing because an attacker must succeed within a small number of allowed attempts before being locked out. If TEE-enforced throttling were bypassed or ineffective, an unthrottled adversary could brute-force the low-entropy password. Thus SVR provides *Resilient-to-Throttled-Guessing* but not *Resilient-to-Unthrottled-Guessing*. Since there is no physical object whose theft would enable recovery, SVR provides *Resilient-to-Theft*. SVR distributes trust across multiple cloud providers using different TEE technologies, so we grant half of *No-Trusted-Third-Party*.

The SVR authors note that their protocol relies on OPRFs that are not post-quantum secure [11]. Since SVR does not use the

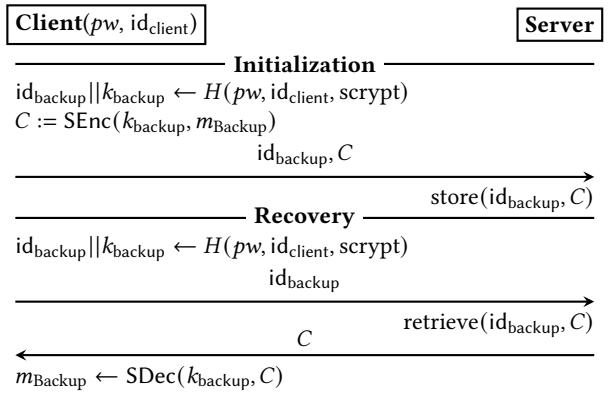


Figure 3: Simplified execution of TS. Transmissions between client and server are secured with TLS.

OPRF as a purely black-box building block, upgrading it to post-quantum security is not straightforward. We therefore do not grant *Post-Quantum-Secure*. SVR also requires a pre-authentication step to associate recovery attempts with a user and prevent attempt exhaustion, so it does not provide *No-Pre-Authentication*. Finally, although trust is distributed across vendors, SVR still depends on the trusted hardware and its setup, and thus does not provide *No-Trusted-Setup*. We are not aware of a formal security proof for SVR, so we do not grant *Security-Proof*.

Regarding storage corruption, SVR stores client-specific state inside the TEEs. If this state leaks across all involved TEEs, an adversary can mount an offline password-guessing attack by testing password guesses locally and checking whether the derived key yields a valid decryption. We therefore grant only half of the *Resilient-to-Storage-Corruption* benefit.

4.4 Threema Safe

Similar to SVR, Threema Safe (TS) is a backup solution that stores a client's account data (i.e., the client's ID, key material, and social graph) but does not include message history or media [29]. Paterson *et al.* analyzed an earlier version of Threema's cryptographic protocols [26]. We supplement their description of Threema Safe with updates from Threema's recent cryptography whitepaper [26, 29]. A simplified execution of TS is depicted in Figure 3.

To set up TS, the client chooses a password with a minimum length of eight characters, which is checked against an offline list of weak passwords. From this password pw and the client's Threema ID (used as a salt), TS derives two 32-byte values (the backup id $\mathsf{id}_{\text{backup}}$ and the backup key k_{backup}) using the hash function scrypt. The client data to be backed up is serialized as a JSON string and encrypted under k_{backup}. The client then uploads the resulting encrypted backup to a backup server over TLS. On the server, the ciphertext is stored under $\mathsf{id}_{\text{backup}}$ as a file identifier.

To restore a TS backup, the client re-derives $\mathsf{id}_{\text{backup}}$ and k_{backup} from the password and Threema ID, uses $\mathsf{id}_{\text{backup}}$ to retrieve the backup from the server, and decrypts the ciphertext using k_{backup}. We remark that Threema allows users and organizations to operate their own backup servers [29]. This is facilitated by the fact that TS does not require specialized hardware such as HSMs. Self-hosting can reduce dependence on a centralized backup service and enables operators to implement defenses such as strict rate limiting, according to their own threat model and operational constraints.

4.4.1 Evaluation. Table 1 in Section 5 summarizes the benefits achieved by TS under our framework.

When TS is used with a low-entropy password, the user must memorize this secret and thus TS provides only half of the *Memorywise-Effortless* benefit. Apart from using the device, the user does not need to manage any physical recovery artifact, so TS provides *Physically-Effortless*. It receives half the *Free-Provider-Choice* benefit as the protocol allows an organization to run their own backup server for its users. Since TS does not require specialized hardware such as HSMs, it provides *No-Special-Hardware-Required*. TS is deployed and available to Threema users and thus provides *Deployed-at-Scale*. Its implementation is published as open source under a free software license, so we grant *Non-Proprietary*.

TS does not incorporate protocol-level rate limiting. Instead, preventing online guessing attacks depends on additional defenses independent of the protocol. Hence, we do not grant *Resilient-to-Throttled-Guessing*. If an adversary can mount an unthrottled guessing attack, the low-entropy password is feasible to brute-force. Hence, TS does not provide *Resilient-to-Unthrottled-Guessing*. Since there is no physical object whose theft would enable recovery, TS provides *Resilient-to-Theft*. At the same time, enforcing the guessing limit requires trusting the backup server (and its operational rate limiting), so TS does not provide *No-Trusted-Third-Party*.

TS is built solely from symmetric and hash-based primitives, which can be configured to remain secure against quantum attackers by choosing appropriate security parameters. We therefore grant *Post-Quantum-Secure*. The scheme does not depend on users authenticating towards the server before retrieving a backup (beyond presenting $\mathsf{id}_{\text{backup}}$ to locate the record) and thus provides *No-Pre-Authentication*. Nevertheless, TS depends on correct server deployment and configuration (in particular for rate limiting), so it does not provide *No-Trusted-Setup*. We are not aware of a formal security proof for TS, and thus do not grant *Security-Proof*.

Regarding storage corruption, the TS server stores encrypted backups indexed by $\mathsf{id}_{\text{backup}}$. If this client ID leaks, an adversary can mount a password-guessing attack by deriving candidate keys from password guesses and checking them against the stored backup (with $\mathsf{id}_{\text{backup}}$ enabling lookup and verification). Since the password is low entropy, this attack is feasible. We therefore grant only half of the *Resilient-to-Storage-Corruption* benefit.

4.5 Key Retrieval with(out) HSM Protection

Faller *et al.* [15] propose three E2EE key-retrieval protocols as alternatives to the WBP under different attacker models. The first assumes an attacker that cannot compromise the HSM, the second assumes an attacker that can compromise the HSM's storage, and the third assumes an attacker capable of full HSM compromise. We name the protocols according to the different attacker models F1, F2, F3, respectively.

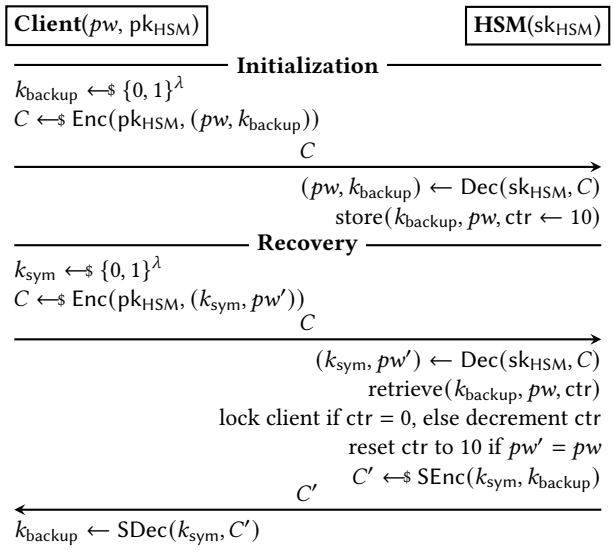


Figure 4: Simplified execution of F1. Messages from the client to the HSM are relayed through an intermediate server, which we omit here.

F1. Under the assumption that the HSM cannot be corrupted and never leaks information, the protocol stores secrets in the clear inside the HSM. A simplified execution of F1 is depicted in Figure 4.

In F1, the client samples a backup encryption key k_{backup} uniformly at random. For initialization, it encrypts it together with a user-chosen password pw under the HSM's public encryption key, and sends it to the HSM. The HSM decrypts the message and stores both key and password in plaintext. The record also contains an attempt counter, initialized to 10, which tracks recovery attempts.

To recover k_{backup}, the client samples a fresh symmetric transport key k_{sym} and sends it alongside a candidate password pw' to the HSM using its public encryption key. The HSM retrieves the client's stored values, and enforces rate limiting: if ctr = 0, it deletes the record; otherwise it decrements the counter and checks whether $pw = pw'$. If the passwords match, the HSM resets the counter and returns k_{backup} encrypted under k_{sym} to the client.

4.5.1 Evaluation of F1. Table 1 in Section 5 summarizes the benefits achieved by F1 under our framework.

The F1 scheme can be used with a low-entropy password. In this case, it provides only half of the *Memorywise-Effortless* benefit since the user must memorize a short secret. As there is no physical recovery artifact to manage beyond the user device, F1 provides *Physically-Effortless*. Since key material must be stored with the central HSM, the scheme does not receive the *Free-Provider-Choice* benefit. The HSM is required to enforce a guessing limit and therefore does not provide *No-Special-Hardware-Required*. Since F1 has not (yet) been deployed in practice, we do not grant *Deployed-at-Scale*. The design is publicly available in [15], thus we grant *Non-Proprietary*.

Due to HSM-enforced rate limiting, an attacker must guess the correct password within ten attempts, so F1 provides *Resilient-to-Throttled-Guessing*. If rate limiting is bypassed or ineffective, an unthrottled attacker can brute-force the low-entropy password. Hence F1 does not provide *Resilient-to-Unthrottled-Guessing*. Since there is no physical object whose theft would enable recovery, F1 provides *Resilient-to-Theft*. However, enforcing the guessing limit relies on a single trusted HSM-based service component, so F1 does not provide *No-Trusted-Third-Party*.

F1 is built from standard cryptographic building blocks for which post-quantum-secure options exist. Thus we grant *Post-Quantum-Secure*. To maintain a per-user attempt counter, F1 requires pre-authentication before recovery and thus does not provide *No-Pre-Authentication*. Since it relies on an HSM, it depends on a trusted setup and therefore does not provide *No-Trusted-Setup*. Faller *et al.* provide a formal security analysis in the UC framework [15], so F1 provides *Security-Proof*.

Finally, F1 stores client secrets (including the backup key and password) in plaintext inside the HSM. Under storage corruption, an adversary can recover the backup key and password immediately, and thus F1 does not provide *Resilient-to-Storage-Corruption*.

F2. Under the assumption that the HSM's storage may be fully leaked to an adversary, F2 hardens F1 by avoiding plaintext password storage and instead storing salted password hashes. A simplified execution of F2 is shown in Figure 5.

During initialization, the HSM generates a fresh public-key pair for the session and sends the public key to the client. The client samples a random backup encryption key k_{backup}, encrypts it and the password pw under the HSM's public key for transport, and sends it to the HSM. The HSM decrypts both values, samples two fresh salts s_1, s_2 to compute a salted hash h and a password-dependent masking value $H(s_2, pw)$, which is used to compute the masked key c. The HSM stores c, h, s_1, s_2 and a counter initialized to 10.

For recovery, the HSM again generates a fresh public-key pair and provides the public key to the client. The client samples a fresh symmetric transport key k_{sym} and sends it alongside a candidate password pw' to the HSM using its public encryption key. The HSM retrieves the client's stored values, and enforces rate limiting: if ctr = 0, it deletes the record; otherwise it decrements the counter and checks whether $H(s_1, pw') = h$. If the equality holds, the HSM resets the counter, unmasks k_{backup}, and returns it encrypted under k_{sym} to the client.

4.5.2 Evaluation of F2. Table 1 in Section 5 summarizes the benefits achieved by F2 under our framework.

Compared to F1, F2 changes only how client state is stored inside the HSM. The key difference is resilience under storage corruption: F2 stores only salted password hashes and a password-masked form of k_{backup}, so a storage leak enables recovery of k_{backup} only after an offline password-guessing attack. Since low-entropy passwords make this additional step feasible, we grant half of *Resilient-to-Storage-Corruption* (whereas F1 receives none). F2 inherits all other benefits from F1.

F3. Under full HSM compromise, the F2 approach becomes strictly weaker: for users who continue to use the service after attestation

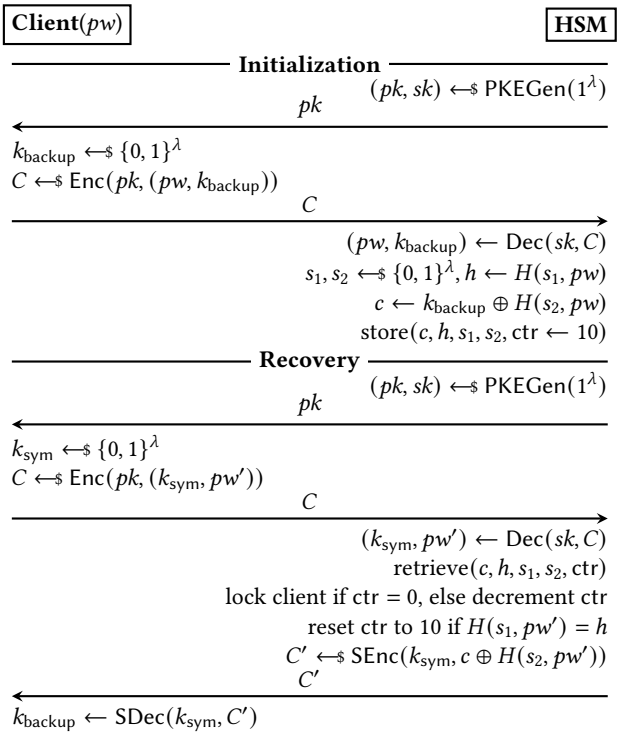


Figure 5: Simplified execution of F2. Messages from the client to the HSM are relayed through an intermediate server, which we omit here.

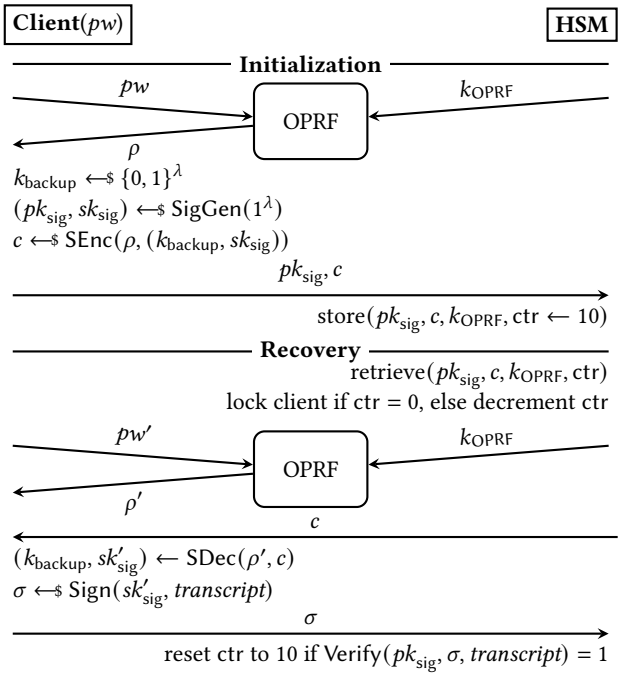


Figure 6: Simplified execution of F3. Messages from the client to the HSM are relayed through an intermediate server, which we omit here. The *transcript* contains previous messages.

key leakage, an adversary can immediately learn plaintext passwords, since it can substitute the public keys under which clients encrypt their inputs. Faller *et al.* propose an OPRF-based protocol F3 to address this weakness. While offline brute-force attacks remain possible, F3 ensures that, even after full HSM compromise, an adversary must still mount an offline attack per user to recover the corresponding backup key, rather than learning passwords immediately. A simplified execution of F3 is depicted in Figure 6.

During initialization, the client and the HSM execute an OPRF in which the client inputs a user-chosen password pw, while the HSM inputs a client-specific OPRF key k_{OPRF}. The client receives the OPRF output ρ that is uses as a key for encryption. The client samples a random backup encryption key k_{backup} and generates a signature key pair $(pk_{\text{sig}}, sk_{\text{sig}})$ for later re-authentication. It then encrypts $(k_{\text{backup}}, sk_{\text{sig}})$ under ρ, obtaining a ciphertext c, which is sent alongside pk_{sig} to the HSM. The HSM stores received values, the key k_{OPRF}, and a counter initialized to ten.

To recover k_{backup}, the client first contacts the HSM, which first enforces rate limiting: If ctr = 0, it deletes the record. Otherwise it decrements the counter. Next, the client and the HSM execute the OPRF protocol again, with the client inputting a candidate password pw' and the HSM using the stored key k_{OPRF}. If and only if $pw = pw'$, the client is able to re-derive ρ and decrypt c sent by the HSM, obtaining the backup key k_{backup} and the secret signing key sk_{sig}. The client can now trigger a reset of the attempt counter by proving successful decryption: it signs a transcript of the prior interaction using sk_{sig}, and the HSM verifies the signature under pk_{sig}. Since producing a valid transcript signature requires recovering sk_{sig}, this serves as evidence that the correct password was used in the OPRF.

4.5.3 Evaluation of F3. Table 1 in Section 5 summarizes the benefits achieved by F3 under our framework.

The key difference to F2 is the robustness of F3 under full HSM compromise. F3 prevents an attacker from immediately learning plaintext passwords. Instead, even after full compromise, the attacker must still carry out an offline password-guessing attack per user to recover k_{backup}. While this does not protect low-entropy passwords, is does have a subtle effect on *high-entropy* ones. Namely, users can be certain that their high-entropy secret cannot be learned by the HSM, even if the HSM's trusted setup was compromised.

In terms of post-quantum migration, F3 uses an OPRF non-black-box [15], which obstructs a generic post-quantum-secure instantiation. Hence, unlike F2, we do not grant *Post-Quantum-Secure*. All other benefits are inherited from F2.

4.6 Juicebox

Juicebox [31] is an open-source secret-recovery protocol developed by Signal co-founder Moxie Marlinspike and collaborators. Similar to SVR, it aims to enable low-entropy, user-chosen passwords pw while mitigating online brute-force attacks via a distributed server-side design. A distinctive feature of Juicebox is that it supports heterogeneous deployments as not all participating servers need to

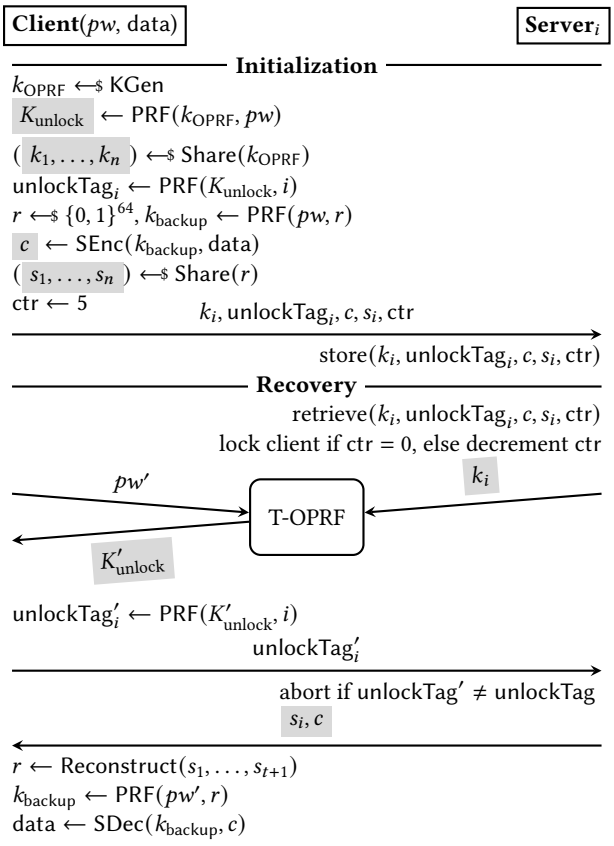


Figure 7: Simplified execution of the Juicebox protocol. Gray boxes highlight values that are protected by a commitment. In Initialization, commitments are created for highlighted values. In Recovery, highlighted values are verified to match the associated values from Initialization using the commitments.

be backed by HSM. Instead, it explicitly considers a mixed deployment of hardware- and software-based servers, which is intended to reduce operational cost and lower the barrier to participation.

Juicebox is a *secret recovery* protocol rather than a pure *key recovery* protocol: the client may store an arbitrary secret data, which may (but need not) be a cryptographic key. This also allows backing up the same secret multiple times. By contrast, many key-recovery schemes assume that the backed-up key is freshly sampled at initialization.[6]

The Juicebox protocol consists of an initialization phase and a recovery phase, illustrated in Figure 7. On a high level, initialization prepares a client-specific record stored across multiple servers. Recovery re-derives the same decryption key in a distributed manner and obtains enough server responses to reconstruct the necessary state.

A simplified execution is as follows. During initialization, the client samples an OPRF key k_{OPRF} uniformly at random and derives a password-dependent value K_{unlock} from k_{OPRF} and the password pw via a PRF. The client then secret-shares k_{OPRF} into $k_1, \ldots, k_n$ with threshold t. For each server i, the client deterministically derives an unlock tag unlockTag_i from K_{unlock} using a deterministic function PRF. Next, the client samples a random value r and derives an encryption key k_{backup} from (pw, r) using a PRF, then encrypts the secret data under k_{backup} using authenticated encryption, obtaining a ciphertext c. The client also secret-shares r into $s_1, \ldots, s_n$ (again with threshold t). Finally, the client selects a recovery-attempt limit[7] ctr (e.g., ctr = 5) and stores, at each server i, the tuple $(k_i, \text{unlockTag}_i, c, s_i, \text{ctr})$ in a client-specific record identified via the scheme's pre-authentication mechanism.

During recovery, each contacted server retrieves the client's record, enforces rate limiting by decrementing the attempt counter ctr, and deletes the record if the counter is exhausted. The client and the servers then evaluate a T-OPRF: the client inputs a password candidate pw', while each server i uses its stored share k_i. Only the client learns the T-OPRF output K'_{unlock}. Using the same deterministic function as in initialization, the client derives per-server unlock tags $\text{unlockTag}'_i$ from K'_{unlock} and sends $\text{unlockTag}'_i$ to each server. Server i checks whether $\text{unlockTag}'_i$ matches the stored unlockTag_i and, if so, returns the corresponding share s_i (and the ciphertext c) to the client. Once the client has obtained at least $t + 1$ valid shares, it reconstructs r, derives k_{backup} from (pw', r) via the PRF, and decrypts c to recover data.

Juicebox additionally includes mechanisms to detect faulty or malicious server behavior. For example, the client forms commitments during initialization (one may think of these as cryptographically protected checksums over selected protocol state) and verifies them during recovery. This allows the client to check that it obtained the expected derived values (e.g., that $K'_{\text{unlock}} = K_{\text{unlock}}$ when the correct password is used) and that servers returned the same ciphertext and corresponding shares as those originally stored. As discussed in [31], these checks can fail only if the underlying commitments can be forged. However, such a forgery is intended to be infeasible under the assumed threshold of honest servers and without guessing the client's password.

4.6.1 Evaluation. Table 1 in Section 5 summarizes the benefits achieved by Juicebox under our framework.

When the Juicebox protocol is used with a low-entropy password, the user must remember a short secret but does not need to manage any paper recovery material. Hence, the scheme provides half of *Memorywise-Effortless* and provides *Physically-Effortless*. While it uses multiple servers the user has no choice where to store key material, thus it does not provide *Free-Provider-Choice*. Juicebox relies on (at least some) HSM-backed servers to enforce rate limiting, and thus it does not provide *No-Special-Hardware-Required*.

While Juicebox has seen deployment in practice (e.g., in the context of Twitter), available reports suggest that it may not be deployed under its intended trust assumptions [23, 30]: if all servers are operated by a single entity, secret sharing no longer distributes trust and does not remove single points of failure. We therefore do not treat this as a large-scale deployment of the scheme in its intended multi-operator setting and do not grant *Deployed-at-Scale*.

[6]Note that a key-recovery scheme can still protect arbitrary secrets by encrypting them under the recovered key using authenticated encryption.

[7]Note that the client must choose this limit to be reasonable low for rate limiting to effectively protect low-entropy passwords.

Juicebox's description is publicly available at [31], so it provides *Non-Proprietary*.

During initialization, the client sets a per-user limit on recovery attempts, which is enforced by the servers during recovery. A throttled adversary must therefore succeed within a bounded number of guesses, so (given the client sets a reasonably low limit) Juicebox provides *Resilient-to-Throttled-Guessing*. If rate limiting is bypassed or ineffective, an unthrottled attacker can brute-force the low-entropy password. Hence the scheme does not provide *Resilient-to-Unthrottled-Guessing*. Since there is no physical object whose theft would enable recovery, Juicebox provides *Resilient-to-Theft*. Trust is distributed across multiple servers rather than a single party, so we grant half of *No-Trusted-Third-Party*.

Juicebox depends on additional cryptographic components (including a 2HashDH-style OPRF and zero-knowledge techniques) [18, 20, 31]. These building blocks are not used in a purely generic, black-box way, so upgrading the protocol to post-quantum security is non-trivial. We therefore do not grant *Post-Quantum-Secure*. The whitepaper also specifies an authentication mechanism to authorize client operations (e.g., initiating recovery attempts) [31], so the scheme does not provide *No-Pre-Authentication*. Moreover, Juicebox's security depends on distributing trust across independently operated servers. Otherwise, secret sharing does not achieve its intended goal [30]. In addition, HSMs introduce trusted-setup assumptions that are largely opaque to users. We therefore do not grant *No-Trusted-Setup*.

The Juicebox whitepaper provides a detailed description but no formal security proof. Thus we do not grant *Security-Proof*. Finally, the client-specific state stored at each server does not directly reveal the decryption key or password. Under storage corruption, an adversary must still mount an offline brute-force attack on the low-entropy password to recover the key. We therefore grant only half of the *Resilient-to-Storage-Corruption* benefit.

4.7 Kintsugi

Kintsugi is an E2EE key-recovery protocol that can also be applied in the instant-messaging setting [22]. Its main design goal is to avoid centralized infrastructure in order to improve metadata privacy and resilience against infrastructure shutdown. To this end, Kintsugi is decentralized and distributes trust across multiple recovery nodes that are intended to be operated by independent parties (analogous to volunteer-run nodes in systems such as Tor). To keep participation feasible, the protocol avoids specialized hardware such as HSMs; Ma *et al.* argue that HSM-based designs can be difficult to deploy and may be cost-prohibitive, pointing to the implementation costs of SVR [22].

By avoiding centralization and specialized hardware, Kintsugi reduces reliance on a single operator. However, this comes with different assumptions and trade-offs. In particular, users must trust that fewer than a threshold of nodes collude. Otherwise, colluding parties may be able to mount offline attacks against the user's password. Moreover, while no single centralized party can shut down the recovery infrastructure, availability depends on a sufficient number of recovery nodes being online. At the cryptographic core, Kintsugi uses a T-OPRF. Ma *et al.* note that choosing the threshold t significantly smaller than the number of recovery nodes n used for registration can improve availability, but also increases the adversary's options when selecting nodes to compromise [22]. A simplified protocol execution is depicted in Figure 8.

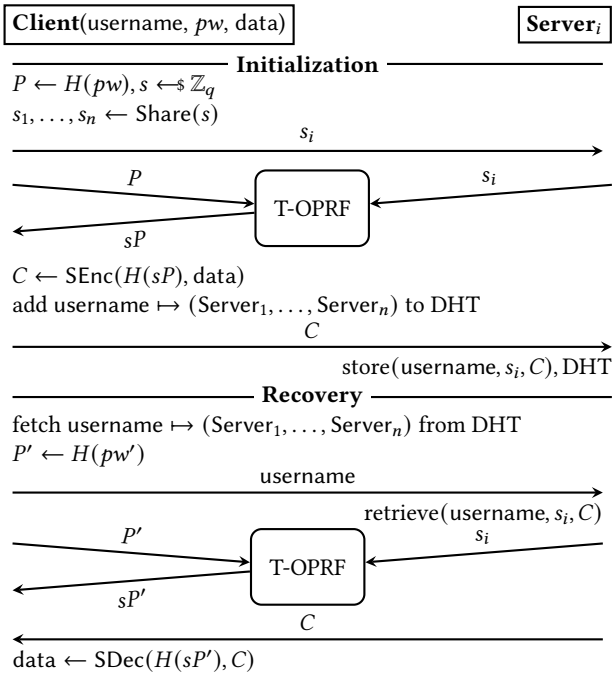


Figure 8: Simplified execution of the Kintsugi protocol, where H is a hash-to-curve function. and DHT is a distributed hash table that stores a list of servers for each username**.**

Every client has a username and a password pw. During initialization, the client first hashes pw to a point P on an elliptic curve. It then samples a random secret $s \in \mathbb{Z}_q$ and splits it into shares $s_1, \ldots, s_n$ using secret sharing with t being the threshold. The client sends each server i its share s_i over an authenticated and encrypted channel. Next, the client and the servers run a T-OPRF in which the client inputs P and each server uses its share as input, leading to the client computing sP.[8] The client uses the hash of sP as a key for authenticated encryption of the backup, producing a ciphertext C. Finally, the client records the set of participating servers in a distributed hash table (DHT), since there is no centralized service that tracks which servers hold state for a given user. Each server stores C together with the username and its share s_i.

For recovery, the client is assumed to remember only its username and a password candidate pw'. It first queries the DHT to retrieve the list of servers associated with its username and contacts them. The client maps pw' to a curve point P' and initiates the T-OPRF with the servers, again using username to reference the record. Each server retrieves its stored share and participates in the T-OPRF. As long as at least $t + 1$ servers respond, the client learns sP'. If and only if $pw = pw'$, it holds that $P = P'$ and the client derives the correct key. After obtaining the ciphertext C from one of the servers, the client can use a hash of sP' to decrypt the backup.

[8] Alternatively, the client could also directly compute sP since it knows both values.

4.7.1 Evaluation. Table 1 in Section 5 summarizes the benefits achieved by Kintsugi under our framework.

The Kintsugi protocol allows the use of a low-entropy user password. In this case, it provides half of *Memorywise-Effortless* since the user must remember a short secret (and their username). As there is no physical object to manage, it provides *Physically-Effortless*. It allows users to make a selection of servers key material is stored across. Therefore it provides *Free-Provider-Choice*. Kintsugi is designed to operate without HSMs and thus provides *No-Special-Hardware-Required*. To the best of our knowledge, Kintsugi has not been adopted at large scale, so we do not grant *Deployed-at-Scale*. Since it is published as open source under the MIT license, it provides *Non-Proprietary*.

Kintsugi does not include any rate limiting on protocol-level. Instead it depends on third party software to prevent online-attacks. Therefore, it does not provide the *Resilient-to-Throttled-Guessing* benefit. If throttling is not implemented, bypassed, or fails, an attacker can brute-force the low-entropy password via online guesses. Hence Kintsugi does not provide *Resilient-to-Unthrottled-Guessing*. Since there is no physical object whose theft would enable recovery, it provides *Resilient-to-Theft*. Trust is distributed across multiple recovery nodes rather than a single service provider, so we grant half of *No-Trusted-Third-Party*.

Kintsugi uses a T-OPRF (TOPPSS [20]) in a non-generic way. As a result, upgrading it to post-quantum security is not straightforward, and we do not grant *Post-Quantum-Secure*. Since Kintsugi does not rely on per-user attempt counters, it does not require a separate pre-authentication step and thus provides *No-Pre-Authentication*. Note that recovery requires the client to provide its username and to look up the relevant server set via a DHT.

Although Kintsugi avoids HSMs, it still relies on assumptions about the recovery-node population and their setup. For example, if a single adversary controls enough nodes (or can influence which nodes a client registers with), the intended trust distribution can be undermined. We therefore do not grant *No-Trusted-Setup*. We are not aware of a formal security proof for Kintsugi, so we do not grant *Security-Proof*.

Regarding storage corruption, Kintsugi servers store client-specific state, including their T-OPRF and secret-sharing inputs. If the storage of at least $t + 1$ servers associated with a user is leaked, an adversary can reconstruct the relevant secret material and mount an offline password-guessing attack. Since the password is low entropy, this additional attack is feasible. We therefore grant only half of the *Resilient-to-Storage-Corruption* benefit.

4.8 Protocol Simplifications

In this subsection we detail our simplifications made to the examined protocols:

4.8.1 Simplifications to the WBP. A detailed description of the WBP is provided in [12]. They also provide a high-level layout of the protocol that we used as a starting point for the simplified execution. We made the following simplifications: While we mention that a WhatsApp server forwards messages between the client and the HSM, we omit the server in Figure 1. We substitute the OPAQUE [21] protocol with a simple OPRF building block. We renamed K^{export} to k_{sym} and K to k_{backup} for consistency across protocol descriptions. In addition, we omitted any notation due to the universal composability framework in which the protocol was modeled.

4.8.2 Simplifications to SVR. The simplified execution of the SVR protocol is based on [11] and the details of the underlying PPSS scheme [19]. We simplified the protocol as follows:

- We omit the authentication process and assume the user is allowed to register and recover a backup.
- We further assume for recovery that a backup of the user has been registered before, as retrieval is impossible otherwise.
- We substitute Shamir's secret sharing with the Share and Reconstruct functions.
- For readability, we abstract the 2HashDH execution into a simple OPRF building block.
- To reflect Signals SVR deployment, we assume $n = 3$ servers are used. This impacts the number of (blinded) secret shares depicted.
- We omit UC modeling artifacts sid and ssid and assume the servers implicitly identify the client-specific values.
- We omit acknowledgment messages as well as error symbols for simplicity.

4.8.3 Simplifications to TS. The TS protocol is described in [29]. We simplified the protocol as follows:

- We omit the scrypt parameters `N, r, p, and dkLen`.
- We omit the `threema-safe-master-key`. Instead we use the values $\mathsf{id}_{\text{backup}}$, k_{backup} derived from it.
- We renamed `threema-safe-encrypted-backup` to C, `threema-safebackup-id` to $\mathsf{id}_{\text{backup}}$, and `Threema ID` to $\mathsf{id}_{\text{client}}$
- We omit the nonce used for encryption.
- We summarized the JSON-serialized user-specific data with m_{Backup}.

4.8.4 Simplifications to the Protocols by Faller et al. The schemes F1, F2, F3 are described in [15]. As they are proven secure in the UC framework, the descriptions contain modeling artifacts as well as auxiliary values used only to identify a specific client among many. We made the following simplifications:

- We omit the setup phase in F1 and assume that the client knows $\mathsf{pk}_{\mathsf{HSM}}$ at the start of the execution already.
- We omit the use of ssid and ID_{C} and assume the HSM can identify the client and its backup without auxiliary information.
- We omit UC-related function calls such as `INITC` and `RECRES`.
- We renamed K to k_{backup}, $\mathsf{sk}_{\mathsf{Enc}}$ to sk, and $\mathsf{pk}_{\mathsf{Enc}}$ to pk
- As we assume recovery works, we omit the transmission of error symbols.

4.8.5 Simplifications to Juicebox. The Juicebox protocol is detailed in the whitepaper [31] and the publicly available source code.[9] In order to explain the parts of the protocol relevant to our comparison, we simplified the protocol as follows:

- We omit the trivial server detection phases "Phase 1 Registration" and "Phase 1 Recovery". We assume that the client

[9] https://github.com/juicebox-systems/juicebox-sdk/blob/main/rust/sdk/src/register.rs

knows which servers to communicate with. We further assume that a sufficient number of servers is available as recovery is impossible otherwise.
- We renamed several variables from the source code for increased readability:
  - `pin` is pw
  - `secret` is data
  - `oprf_private_key` is k_{OPRF}
  - `oprf_private_key_share` is k_i
  - `unlock_key_tag` is unlockTag_i
  - `unlock_key` is K_{unlock}
  - `realm.id` is i
  - `scalar` is r
  - `encryption_key` is k_{backup}
  - `encryption_key_scalar_share` is s_i
  - `encrypted_secret` is c
- In place of `access_key` and `encryption_key_seed` which are both derived from `pin`, we simply use pw directly.
- Instead of using blinded OPRF inputs and outputs we simplify them into a T-OPRF building block.
- We omit the parameters `configuration` and `version` due to the availability assumptions mentioned above.
- For readability, we highlight the values that are verified using a commitment. We omit how the individual commitments are created.
- We substitute Shamir's secret sharing with the Share and Reconstruct functions.

4.8.6 Simplifications to Kintsugi. Kintsugi is detailed in [22]. In our simplified protocol execution we substitute Shamir's secret sharing with the Share and Reconstruct functions. Additionally, we use a generic OPRF building block in place of the TOPPSS [20] scheme. We also omit the details of the DHT implementation, and made an additional hash of the elliptic curve point sP explicit.

4.9 The Impact of High-Entropy Secrets

All of the above schemes can also be used with a high-entropy secret instead. In line with our threat model, we treat "high entropy" as a property of the secret's distribution, i.e., the secret is sampled uniformly. If the user chooses the secret, the analysis reverts to the low-entropy setting. We find this reasonable, since humans are unlikely to reliably generate and manage multiple independent high-entropy secrets across services without falling back to reuse or recognizable structure. In particular, if a user self-selects one "high-entropy" secret and reuses it across services with only low-entropy variations (e.g., appending numbers), then a leak of any one variant can enable efficient guessing of the others, effectively negating the security advantages of high-entropy secrets.

Consequently, we assume this secret is non-memorizable and stored externally on a physical document, analogously to recovery codes. Switching from a low-entropy password to a high-entropy secret changes the benefits of the schemes as follows.

First, the usability and theft-related benefits shift. Since the user no longer has to memorize a secret, schemes attain the full *Memorywise-Effortless* benefit. At the same time, storing recovery material reintroduces physical effort and a theft surface. Consequently, schemes lose *Physically-Effortless* and *Resilient-to-Theft*.

Second, high entropy makes brute-force guessing infeasible. As a result, schemes attain *Resilient-to-Throttled-Guessing* and *Resilient-to-Unthrottled-Guessing* even if any server-side rate limiting fails. In other words, the confidentiality of the recovered key is no longer dependent on throttling, but on the entropy of the user secret.

Third, for schemes where the server or HSM only ever stores ciphertexts or password-blinded values, high entropy removes the practical relevance of offline guessing after leakage. Under this variant, WBP+, SVR+, TS+, F3+, Juicebox+, and Kintsugi+ achieve *No-Trusted-Third-Party* and *No-Trusted-Setup* in our model: compromising the service infrastructure does not yield a feasible path to recover the backup key via guessing. By contrast, F1+ and F2+ still provide the HSM with unblinded material in the client record and therefore continue to require trust in the HSM and its setup.

Finally, assuming storage corruption occurs after initialization, all schemes except F1+ attain *Resilient-to-Storage-Corruption*. In these schemes, an attacker would still need to mount an additional offline attack, but this becomes infeasible given a high-entropy secret. In F1+, storage corruption trivially reveals the protected values and thus the scheme remains insecure in this setting. All remaining benefits are unaffected by switching from low-entropy passwords to high-entropy user secrets.

5 Discussion

The evaluation results are depicted in Table 1. We denote the variant where a scheme is used with a high-entropy user secret by appending a "+" to the protocol name (e.g., WBP+).

Usability. For usability, we observe a trade-off between being *Memorywise-Effortless* and being *Physically-Effortless*. Schemes that avoid physical recovery material (e.g., paper recovery codes) typically rely on a memorizable low-entropy secret, while schemes that maximize resistance to guessing do so by requiring the user to handle and protect high-entropy recovery material. Interestingly, a similar tension between cognitive and physical effort was also identified by Bonneau *et al.* for web authentication schemes [8]. In the rare-retrieval setting, this trade-off is particularly interesting: The recovery mechanism is meant to be used rarely, so added effort may be acceptable, but only if it does not undermine reliability when recovery is actually needed.

Beyond recovery codes, only the decentralized Kintsugi protocol provides *Free-Provider-Choice* in the strict sense, since clients can select a set of independent recovery nodes. Threema Safe does not give end users such choice by default, but it supports self-hosting. This still matters in practice. Self-hosting shifts operational control and data sovereignty to the user's organization. For example, a company can run its own TS server to keep backup metadata and availability guarantees under its own administrative domain. In this sense, self-hosting reduces dependence on external hosting providers. However, it does not eliminate trust assumptions. Rather, it relocates them from a third-party operator to the self-hosting operator's own security and operational practices.

Deployability. Looking at deployability, most examined designs rely on specialized hardware such as HSMs or TEEs. In effect, many providers aim to "outsource" the user's trust in the provider toward secure hardware components. While this can strengthen certain

Table 1: Comparison of key-retrieval schemes.

				Usability			Deployability			Security									
Category	Scheme	Described in section	Reference	*Memorywise-Effortless*	*Physically-Effortless*	*Free-Provider-Choice*	*No-Special-Hardware-Required*	*Deployed-at-Scale*	*Non-Proprietary*	*Resilient-to-Throttled-Guessing*	*Resilient-to-Unthrottled-Guessing*	*Resilient-to-Theft*	*No-Trusted-Third-Party*	*Post-Quantum-Secure*	*No-Pre-Authentication*	*No-Trusted-Setup*	*Security-Proof*	*Resilient-to-Storage-Corruption*
High Entropy	Recovery Codes	4.1		●	○	●	●	●	●	●	●	○	●	●	●	●	●	●
Low Entropy	WBP	4.2	[12]	◎	●	○	○	●	○	●	○	●	○	○	○	○	●	◎
	SVR	4.3	[11]	◎	●	○	○	●	●	●	○	●	◎	○	○	○	○	◎
	TS	4.4	[29]	◎	●	◎	●	●	●	○	○	●	○	●	●	○	○	◎
	F1	4.5	[15]	◎	●	○	○	○	●	●	○	●	○	●	○	○	●	○
	F2	4.5	[15]	◎	●	○	○	○	●	●	○	●	○	●	○	○	●	◎
	F3	4.5	[15]	◎	●	○	○	○	●	●	○	●	○	○	○	○	●	◎
	Juicebox	4.6	[31]	◎	●	○	○	○	●	●	○	●	◎	○	○	○	○	◎
	Kintsugi	4.7	[22]	◎	●	●	●	○	●	○	○	●	◎	○	●	○	○	◎
High Entropy	WBP+	4.2	[12]	●	○	○	○	●	○	●	●	○	●	○	○	●	●	●
	SVR+	4.3	[11]	●	○	○	○	●	●	●	●	○	●	○	○	●	○	●
	TS+	4.4	[29]	●	○	◎	●	●	●	●	●	○	●	●	●	●	○	●
	F1+	4.5	[15]	●	○	○	○	○	●	●	●	○	○	●	○	○	●	○
	F2+	4.5	[15]	●	○	○	○	○	●	●	●	○	○	●	○	○	●	●
	F3+	4.5	[15]	●	○	○	○	○	●	●	●	○	●	○	○	●	●	●
	Juicebox+	4.6	[31]	●	○	○	○	○	●	●	●	○	●	○	○	●	○	●
	Kintsugi+	4.7	[22]	●	○	●	●	○	●	●	●	○	●	○	●	●	○	●

●= offers the full benefit; ◎= offers half the benefit; ○= does not offer the benefit

defenses (e.g., enforcing a guessing limit), it also changes the trust landscape: security of hardware components, trusted setup ceremonies, and vendor ecosystems become trust anchors. We also observe that only a minority of schemes are deployed at scale, while most proposals remain research designs. On the positive side, essentially all schemes except the WBP are non-proprietary, which fosters independent auditing and follow-up research.

Security. From a security perspective, all schemes in our comparison either enforce rate limiting at the protocol level or recommend implementing via additional measures. This is essential for any scheme that permits low-entropy, user-chosen secrets. However, it also highlights a structural limitation. If rate limiting fails (e.g., due to misconfiguration, compromise, or a break of the trusted component) then none of the low-entropy variants is resilient to unthrottled online guessing. In contrast, using a high-entropy secret is an effective mitigation that restores resilience against unthrottled guessing, even if throttling fails.

We observe that usability and security are not independent. Physical recovery material is burdensome, but it is precisely that material which creates a theft surface. In contrast, schemes that eliminate physical artifacts can improve theft resistance but often do so by relying on low-entropy secrets whose practical security depends heavily on throttling.

Distribution of trust across multiple servers appears only in a small subset of schemes. While this can reduce the risk of a single compromised server immediately breaking recovery, it does not eliminate trust when low-entropy secrets are used. More precisely: with low-entropy secrets, even threshold designs still require trusting that (i) rate limiting holds and (ii) the adversary cannot compromise enough servers (or their setup) to enable offline or online guessing at scale. A *full* removal of third-party trust is only plausible when the recovery secret has high entropy and the service stores no plaintext (or easily verifiable) password-dependent state that makes guessing feasible after leakage.

Regarding post-quantum considerations, we observe a pattern consistent with [15]. Simpler schemes built from standard primitives tend to be quantum-safe by default or easier to upgrade, while designs that rely on (T-)OPRFs and related advanced cryptographic primitives are currently difficult to upgrade. Ongoing research addresses this challenge [14], but practical post-quantum migration remains non-trivial for many deployed constructions.

Finally, only a few schemes avoid additional pre-authentication. This is largely explained by the widespread use of per-user recovery attempt counters, which must be protected against denial-of-service attacks by malicious clients who try to exhaust another user's

attempts. For our comparison, we assume access to an idealized pre-authentication mechanism. In practice, dependence on additional authentication introduces its own attack surface for impersonation.

Across all schemes that store client-specific recovery state, storage corruption enables offline attacks in principle. Once an attacker obtains the relevant state, they can often test password candidates locally. Our results indicate that high-entropy secrets are an effective mitigation that can render such offline guessing infeasible. Importantly, this mitigation only applies if the server/HSM does not store the secret in plaintext.

Additional Observations. Beyond what is directly captured by the benefit table, we highlight three observations that matter specifically in the rare-retrieval setting.

First, the push toward low-entropy secrets combined with centralized trusted components or opaque trusted setup procedures creates a risk that is easy to underestimate in formal models. Such systems can become attractive high-value targets for large-scale adversaries. Compromise or subversion of a single infrastructure component can have severe consequences. In light of historic political pressure on E2EE-adoption [7] and cryptographic control policies [1–3] this should not be overlooked. This risk should be treated as a design concern rather than a peripheral detail.

Second, physically stored recovery codes shift the main attack surface from a scalable "break once, harvest many" infrastructure compromise to per-user attacks. While paper recovery material introduces theft risk at the individual level, it is inherently difficult to steal at scale. An attacker typically needs physical proximity, targeted access, or successful social engineering for each user. Even large-scale adversaries can rarely turn this into a fully automated pipeline comparable to compromising a centralized recovery service or a widely deployed secure-hardware ecosystem. In this sense, recovery codes trade a catastrophic single point of failure for effortful attacks whose cost grows with the number of victims.

Third, rare retrieval interacts uncomfortably with human behavior. When systems ask users to rely on short, memorizable secrets, effective security can be dominated by password hygiene and reuse. For context, a Google/Harris poll reports that only 35% of respondents use a different password for all accounts, while 52% reuse the same password for multiple accounts and 13% reuse the same password for all accounts.[10] In such settings, an attacker may succeed in very few guesses if the recovery secret is reused or derived from other breached credentials, which weakens the practical value of throttling. Note that this even holds for a high-entropy password chosen by the user. This motivates treating reuse-driven risks as a practical concern, even if many recovery protocols implicitly assume non-reused secrets. A possible mitigation would be to have the application chose the high-entropy secret in rare-retrieval protocols, e.g., via a recovery code.

Future Research. Our results point to several broader research directions that are largely orthogonal to "yet another stronger protocol." The common thread is that rare retrieval sits at the intersection of cryptography, systems, and human behavior, and security outcomes are often determined by factors that are easy to abstract away in security models.

[10] https://services.google.com/fh/files/blogs/google_security_infographic.pdf

Even if recovery secrets are meant to be service-specific, users often reuse patterns across services. This makes large-scale guessing and reuse-driven attacks a practical concern. A promising direction is to study how often recovery secrets are reused or correlated with other secrets, and to develop interventions that reduce reuse without imposing unrealistic cognitive burden.

Our comparison highlights that many modern designs shift trust toward specialized infrastructure. This creates attractive targets for large-scale adversaries. Future research should explore designs that minimize such single points of failure, make trust distribution meaningful in practice (e.g., via independent operators) without pushing an unbearable burden onto users.

Overall, these directions emphasize that improving rare-retrieval security is not only about stronger cryptographic primitives. It also requires understanding human factors so that designs do not inadvertently trade one class of risk for another, or create high-value points of failure that large-scale adversaries can exploit.

6 Conclusion

In this work, we introduced benefit-based framework to evaluate E2EE key-retrieval schemes along usability, deployability, and security dimensions, and applied it to recovery codes, real-world messenger deployments, and proposals from the literature.

Our comparison highlights a recurring tension. Schemes that aim to support low-entropy, memorizable secrets typically rely on throttling mechanisms and additional trust assumptions, often involving specialized hardware or opaque setup procedures. While these choices can protect against online guessing in the intended setting, they also introduce new attack surfaces and potential single points of failure that are attractive to large-scale adversaries. In contrast, using a high-entropy secret can eliminate entire classes of guessing attacks and reduce reliance on third parties, but it reintroduces physical effort and theft risk, reflecting a fundamental usability-security trade-off in rare retrieval.

Overall, our results show that progress in this field should not be measured solely by strengthening cryptographic properties in isolation. Instead, E2EE backup recovery should be evaluated by the real-world security it provides. We hope our framework and findings help practitioners and researchers make these trade-offs explicit, and motivate future work on recovery mechanisms that align cryptographic design with realistic large-scale threat models.

References

[1] Harold Abelson, Ross J. Anderson, Steven M. Bellovin, Josh Benaloh, Matt Blaze, Jon Callas, Whitfield Diffie, Susan Landau, Peter G. Neumann, Ronald L. Rivest, Jeffrey I. Schiller, Bruce Schneier, Vanessa Teague, and Carmela Troncoso. 2024. Bugs in our pockets: the risks of client-side scanning. *J. Cybersecur.* 10, 1 (2024). doi:10.1093/CYBSEC/TYAD020

[2] Harold Abelson, Ross J. Anderson, Steven M. Bellovin, Josh Benaloh, Matt Blaze, Whitfield Diffie, John Gilmore, Matthew Green, Susan Landau, Peter G. Neumann, Ronald L. Rivest, Jeffrey I. Schiller, Bruce Schneier, Michael A. Specter, and Daniel J. Weitzner. 2015. Keys under doormats. *Commun. ACM* 58, 10 (2015), 24–26. doi:10.1145/2814825

[3] Hal Abelson, Ross J. Anderson, Steven M. Bellovin, Josh Benaloh, Matt Blaze, Whitfield Diffie, John Gilmore, Peter G. Neumann, Ronald L. Rivest, Jeffrey I. Schiller, and Bruce Schneier. 1997. The risks of key recovery, key escrow, and trusted third-party encryption. *World Wide Web J.* 2 (1997), 241–257.

[4] Minh Hieu Nguyen Ba, Jacob Bennett, Michael Gallagher, and Suman Bhunia. 2021. A Case Study of Credential Stuffing Attack: Canva Data Breach. In *International Conference on Computational Science and Computational Intelligence, CSCI 2021, Las Vegas, NV, USA, December 15-17, 2021*. IEEE, 735–740.

doi:10.1109/CSCI54926.2021.00187

[5] Ero Balsa, Helen Nissenbaum, and Sunoo Park. 2022. Cryptography, Trust and Privacy: It's Complicated. In *Proceedings of the 2022 Symposium on Computer Science and Law, CSLAW 2022, Washington DC, USA, November 1-2, 2022*, Daniel J. Weitzner, Joan Feigenbaum, and Christopher S. Yoo (Eds.). ACM, 167–179. doi:10.1145/3511265.3550443

[6] Daniel J. Bernstein and Tanja Lange. 2017. Post-quantum cryptography. *Nat.* 549, 7671 (2017), 188–194. doi:10.1038/NATURE23461

[7] Jenny Blessing, Daniel Hugenroth, Ross J. Anderson, and Alastair R. Beresford. 2025. SoK: Web Authentication and Recovery in the Age of End-to-End Encryption. *Proc. Priv. Enhancing Technol.* 2025, 3 (2025), 560–589. doi:10.56553/POPETS-2025-0113

[8] Joseph Bonneau, Cormac Herley, Paul C. van Oorschot, and Frank Stajano. 2012. The Quest to Replace Passwords: A Framework for Comparative Evaluation of Web Authentication Schemes. In *IEEE Symposium on Security and Privacy, SP 2012, 21-23 May 2012, San Francisco, California, USA*. IEEE Computer Society, 553–567. doi:10.1109/SP.2012.44

[9] Ran Canetti. 2001. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*. IEEE Computer Society, 136–145. doi:10.1109/SFCS.2001.959888

[10] George Coker, Joshua D. Guttman, Peter A. Loscocco, Amy L. Herzog, Jonathan K. Millen, Brian O'Hanlon, John D. Ramsdell, Ariel Segall, Justin Sheehy, and Brian T. Sniffen. 2011. Principles of remote attestation. *Int. J. Inf. Sec.* 10, 2 (2011), 63–81. doi:10.1007/S10207-011-0124-7

[11] Graeme Connell, Vivian Fang, Rolfe Schmidt, Emma Dauterman, and Raluca Ada Popa. 2024. Secret Key Recovery in a Global-Scale End-to-End Encryption System. In *18th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2024, Santa Clara, CA, USA, July 10-12, 2024*, Ada Gavrilovska and Douglas B. Terry (Eds.). USENIX Association, 703–719. https://www.usenix.org/conference/osdi24/presentation/connell

[12] Gareth T. Davies, Sebastian H. Faller, Kai Gellert, Tobias Handirk, Julia Hesse, Máté Horváth, and Tibor Jager. 2023. Security Analysis of the WhatsApp End-to-End Encrypted Backup Protocol. In *Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, 2023, Proceedings, Part IV (Lecture Notes in Computer Science, Vol. 14084)*, Helena Handschuh and Anna Lysyanskaya (Eds.). Springer, 330–361. doi:10.1007/978-3-031-38551-3_11

[13] Yihe Duan, Ding Wang, and Yanduo Fu. 2025. Security Analysis of Master-Password-Protected Password Management Protocols. In *IEEE Symposium on Security and Privacy, SP 2025, San Francisco, CA, USA, May 12-15, 2025*, Marina Blanton, William Enck, and Cristina Nita-Rotaru (Eds.). IEEE, 701–719. doi:10.1109/SP61157.2025.00050

[14] Sebastian Faller, Marc Fischlin, Julius Hardt, and Julia Hesse. 2025. Combining Oblivious Pseudorandom Functions. Cryptology ePrint Archive, Paper 2025/1084. https://eprint.iacr.org/2025/1084

[15] Sebastian H. Faller, Tobias Handirk, Julia Hesse, Máté Horváth, and Anja Lehmann. 2024. Password-Protected Key Retrieval with(out) HSM Protection. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security, CCS 2024, Salt Lake City, UT, USA, October 14-18, 2024*, Bo Luo, Xiaojing Liao, Jun Xu, Engin Kirda, and David Lie (Eds.). ACM, 2445–2459. doi:10.1145/3658644.3690358

[16] Lov K. Grover. 1996. A Fast Quantum Mechanical Algorithm for Database Search. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22-24, 1996*, Gary L. Miller (Ed.). ACM, 212–219. doi:10.1145/237814.237866

[17] Sandra Höltervennhoff, Noah Wöhler, Arne Möhle, Marten Oltrogge, Yasemin Acar, Oliver Wiese, and Sascha Fahl. 2024. A mixed-methods study on user experiences and challenges of recovery codes for an end-to-end encrypted service. In *Proceedings of the 33rd USENIX Conference on Security Symposium* (Philadelphia, PA, USA) *(SEC '24)*. USENIX Association, USA, Article 406, 18 pages.

[18] Stanislaw Jarecki, Aggelos Kiayias, and Hugo Krawczyk. 2014. Round-Optimal Password-Protected Secret Sharing and T-PAKE in the Password-Only Model. In *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014, Proceedings, Part II (Lecture Notes in Computer Science, Vol. 8874)*, Palash Sarkar and Tetsu Iwata (Eds.). Springer, 233–253. doi:10.1007/978-3-662-45608-8_13

[19] Stanislaw Jarecki, Aggelos Kiayias, Hugo Krawczyk, and Jiayu Xu. 2016. Highly-Efficient and Composable Password-Protected Secret Sharing (Or: How to Protect Your Bitcoin Wallet Online). In *IEEE European Symposium on Security and Privacy, EuroS&P 2016, Saarbrücken, Germany, March 21-24, 2016*. IEEE, 276–291. doi:10.1109/EUROSP.2016.30

[20] Stanislaw Jarecki, Aggelos Kiayias, Hugo Krawczyk, and Jiayu Xu. 2017. TOPPSS: Cost-Minimal Password-Protected Secret Sharing Based on Threshold OPRF. In *Applied Cryptography and Network Security - 15th International Conference, ACNS 2017, Kanazawa, Japan, July 10-12, 2017, Proceedings (Lecture Notes in Computer Science, Vol. 10355)*, Dieter Gollmann, Atsuko Miyaji, and Hiroaki Kikuchi (Eds.). Springer, 39–58. doi:10.1007/978-3-319-61204-1_3

[21] Stanislaw Jarecki, Hugo Krawczyk, and Jiayu Xu. 2018. OPAQUE: An Asymmetric PAKE Protocol Secure Against Pre-computation Attacks. In *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part III (Lecture Notes in Computer Science, Vol. 10822)*, Jesper Buus Nielsen and Vincent Rijmen (Eds.). Springer, 456–486. doi:10.1007/978-3-319-78372-7_15

[22] Emilie Ma and Martin Kleppmann. 2025. Kintsugi: Decentralized E2EE Key Recovery. *CoRR* abs/2507.21122 (2025). arXiv:2507.21122 doi:10.48550/ARXIV.2507.21122

[23] Matthew Green. 2025. A bit more on Twitter/X's new encrypted messaging. https://blog.cryptographyengineering.com/2025/06/09/a-bit-more-on-twitter-xs-new-encrypted-messaging/

[24] Meta. 2023. The Labyrinth Encrypted Message Storage Protocol. https://engineering.fb.com/wp-content/uploads/2023/12/TheLabyrinthEncryptedMessageStorageProtocol_12-6-2023.pdf

[25] National Institute of Standards and Technology. 2024. *Cryptographic Algorithms and Key Sizes for Personal Identity Verification*. Technical Report Federal Information Processing Standards Publications (FIPS) 140-2, Change Notice 2 December 03, 2002. U.S. Department of Commerce, Washington, D.C. doi:10.6028/NIST.SP.800-78-5

[26] Kenneth G. Paterson, Matteo Scarlata, and Kien Tuong Truong. 2023. Three Lessons From Threema: Analysis of a Secure Messenger. In *32nd USENIX Security Symposium, USENIX Security 2023, Anaheim, CA, USA, August 9-11, 2023*, Joseph A. Calandrino and Carmela Troncoso (Eds.). USENIX Association, 1289–1306. https://www.usenix.org/conference/usenixsecurity23/presentation/paterson

[27] Moritz Schneider, Ramya Jayaram Masti, Shweta Shinde, Srdjan Capkun, and Ronald Perez. 2022. SoK: Hardware-supported Trusted Execution Environments. *CoRR* abs/2205.12742 (2022). arXiv:2205.12742 doi:10.48550/ARXIV.2205.12742

[28] Richard Shay, Patrick Gage Kelley, Saranga Komanduri, Michelle L. Mazurek, Blase Ur, Timothy Vidas, Lujo Bauer, Nicolas Christin, and Lorrie Faith Cranor. 2012. Correct horse battery staple: exploring the usability of system-assigned passphrases. In *Symposium On Usable Privacy and Security, SOUPS '12, Washington, DC, USA - July 11 - 13, 2012*, Lorrie Faith Cranor (Ed.). ACM, 7. doi:10.1145/2335356.2335366

[29] Threema. 2025. Cryptography Whitepaper. https://threema.com/assets/documents/cryptography_whitepaper.pdf

[30] Nora Trapp. 2025. Don't Put All Your Juice in One Box. https://juicebox.xyz/blog/dont-put-all-your-juice-in-one-box

[31] Nora Trapp and Diego Ongaro. 2025. Juicebox Protocol: Distributed Storage and Recovery of Secrets Using Simple PIN Authentication. *IACR Cryptol. ePrint Arch.* (2025), 348. https://eprint.iacr.org/2025/348

[32] WhatsApp. 2021. Security of End-To-End Encrypted Backups. https://www.whatsapp.com/security/WhatsApp_Security_Encrypted_Backups_Whitepaper.pdf