Threshold UX: Making Quorum, Recovery, and Shared Control Normal Product Experiences

Invisible Threshold Cryptography for Approval Flows, Social Recovery, and Progressive Security on Lux Network

Zach Kelling
Lux Industries

2025

Abstract

Threshold cryptography is one of the most powerful primitives in applied cryptography: t-of-n signature schemes, secret sharing, and multi-party computation enable security properties—recovery without seed phrases, shared control without a single admin, regulatory compliance without centralized escrow—that no single-key system can provide. Yet threshold cryptography remains almost entirely absent from consumer products. The reason is not technical. The primitives are mature. The reason is UX: threshold protocols are presented to users as cryptographic ceremonies involving share distribution, quorum assembly, and reconstruction rituals that make no sense outside a cryptographer's mental model. This paper presents Threshold UX—a set of design patterns that embed threshold cryptography into normal product experiences on Lux Network. We show how t-of-n threshold signatures (FROST on the M-Chain) map to approval flows ("2 of 3 team members must approve"), social recovery ("any 3 of your 5 trusted contacts can restore your account"), shared workspaces (threshold access control over collaborative documents), treasury management (multi-party spend authorization), regulated access (compliance-gated reveal), and progressive security (start simple, add guardians as stakes increase). We formalize each pattern as a composition of T-Chain and M-Chain operations, prove security properties under standard threshold assumptions, and define the UX interaction model that hides cryptographic complexity behind familiar product metaphors.

1 Introduction

Threshold cryptography solves a fundamental problem in digital security: how to distribute trust so that no single party—no single device, no single employee, no single server—can unilaterally access, authorize, or destroy a protected resource. The theoretical foundations are decades old. Shamir's secret sharing [1] dates to 1979. Threshold signature schemes have been refined through Gennaro and Goldfeder [2], FROST [3], and CGGMP21 [4]. Fully homomorphic encryption enables threshold policy evaluation without decryption [5].

Despite this maturity, threshold cryptography has near-zero penetration in consumer software. The handful of deployments—multisig wallets in cryptocurrency, HSM quorum policies in enterprise key management—are used exclusively by technical specialists who understand the underlying cryptographic model. No mainstream application presents threshold operations as a natural product experience.

The gap is not technical. It is a design failure. Threshold cryptography is presented to users in its own terms: "distribute n shares, assemble t shares, reconstruct the secret." These terms mean nothing to a user who wants to share a document with their team, recover their account after losing their phone, or require two signatures on a large payment. The cryptographic ceremony is visible, confusing, and intimidating.

This paper closes the gap by defining Threshold UX: a mapping from threshold cryptographic primitives to familiar product interaction patterns on Lux Network. The key insight is that threshold operations already have natural product analogues:

- A t-of-n threshold signature is an *approval flow*: "2 of 3 team members must approve this action."
- Shamir secret sharing is *social recovery*: "Any 3 of your 5 trusted contacts can help you regain access."
- Threshold access control is a *shared workspace*: "Everyone on the team can read; edits require approval."
- Threshold spend authorization is *treasury management*: "Large payments need two signers."

- Threshold reveal is *regulated access*: "A compliance officer must co-approve before viewing PII."

The user never sees shares, quorums, or reconstruction. They see approvals, guardians, team permissions, and spending limits. The cryptography is real. The ceremony is gone.

Section 2 defines the threshold primitives available on Lux. Section 3 maps these to approval flows. Section 4 addresses social recovery. Section 5 covers shared workspaces. Section 6 formalizes treasury management. Section 7 describes regulated access. Section 8 introduces progressive security. Section 9 defines UX interaction patterns. Section 10 proves security properties. Section 11 concludes.

2 Threshold Primitives on Lux

Lux Network provides threshold cryptography through two dedicated chains:

1. **T-Chain (Threshold Chain).** Manages threshold policies, TFHE-encrypted policy evaluation, and CKKS confidential compute. The T-Chain stores encrypted policy conditions and evaluates them homomorphically, producing a binary approve/deny decision without revealing the policy inputs to any single party.

2. **M-Chain (MPC Chain).** Executes multi-party threshold signatures via FROST (for Schnorr/Ed25519) and CGGMP21 (for ECDSA). The M-Chain coordinates distributed key generation (DKG), signing rounds, and key refresh without any party learning the full private key.

2.1 FROST Threshold Signatures

FROST [3] is a round-optimized Schnorr threshold signature scheme. In the Lux deployment, FROST operates as follows:

Definition 1 (FROST Configuration)**.** *A FROST configuration* $\Phi = (t, n, \{P_1, \ldots, P_n\}, \mathsf{pk})$ *consists of a threshold* t*, a total participant count* n*, a set of participants* P_i *each holding a key share* s_i*, and the aggregate public key* pk*. A valid signature requires the cooperation of any* t *of the* n *participants.*

The signing protocol proceeds in two rounds:

Algorithm 1 FROST Threshold Signing (simplified)

Require: Message m, threshold t, participants $\{P_{i_1}, \ldots, P_{i_t}\}$

1: **Round 1 (Commitment):** Each P_{i_j} generates nonce pair (d_{i_j}, e_{i_j}) and broadcasts commitments (D_{i_j}, E_{i_j})

2: **Round 2 (Response):** Each P_{i_j} computes partial signature z_{i_j} using Lagrange interpolation coefficients

3: **Aggregation:** Coordinator computes $\sigma = (R, z)$ where $R = \prod D_{i_j}^{\rho_{i_j}} \cdot E_{i_j}$ and $z = \sum z_{i_j}$

4: **return** σ (standard Schnorr signature, verifiable with pk)

The output is an ordinary Schnorr signature. No verifier can distinguish a FROST threshold signature from a single-signer Schnorr signature. This property is essential for Threshold UX: the cryptographic machinery is invisible to anyone not participating in the signing ceremony.

2.2 TFHE Policy Evaluation

The T-Chain evaluates access control policies over encrypted inputs using TFHE (Torus Fully Homomorphic Encryption) [6]. A policy π is a Boolean circuit that takes encrypted attributes as input and produces an encrypted Boolean decision:

$$\mathsf{Eval}(\pi, \mathsf{Enc}(a_1), \ldots, \mathsf{Enc}(a_k)) = \mathsf{Enc}(\pi(a_1, \ldots, a_k)) \tag{1}$$

The decision is threshold-decrypted by t-of-n T-Chain validators, revealing only the binary result (approve/deny) without exposing the input attributes or the policy logic.

3 Approval Flows

An approval flow is the most direct product mapping of threshold signatures. The user mental model is: "This action requires approval from t of n designated approvers before it executes."

3.1 Approval Flow Architecture

Definition 2 (Approval Flow)**.** *An approval flow* $\mathcal{A} = (a, t, n, \{A_1, \ldots, A_n\}, \tau, \mathsf{esc})$ *consists of:*

- a*: the action to be approved (transaction, access request, configuration change)*
- t, n*: the threshold and total approver count*
- $\{A_1, \ldots, A_n\}$*: the set of designated approvers*
- τ*: the timeout (maximum time to collect* t *approvals)*
- esc*: the escalation policy (what happens if* τ *expires without* t *approvals)*

When a user initiates an action that requires approval, the system:

1. Creates a pending action record on the T-Chain with the action payload and approval metadata.
2. Sends push notifications to all n approvers: "[User] requests approval for [action description]."
3. Each approver reviews the action and either approves (producing a FROST partial signature) or rejects (recording a rejection on-chain).
4. When t partial signatures are collected, the M-Chain aggregates them into a complete threshold signature, which authorizes the action.
5. The action executes with the threshold signature as its authorization proof.

The approver sees: "Approve this payment of $5,000 to Acme Corp? [Approve] [Reject]." They do not see: "Please contribute your Shamir share to the threshold ECDSA signing ceremony."

3.2 Timeout and Escalation

Real approval flows must handle timeouts. If approvers are unavailable, the action cannot remain pending indefinitely. The escalation policy esc defines the fallback:

1. **Auto-deny.** If τ expires, the action is denied. This is the default for high-security actions.
2. **Escalation.** If τ expires, the approval request escalates to a secondary approver set $\{A'_1, \ldots, A'_{n'}\}$ with a new timeout τ'.
3. **Threshold reduction.** If τ expires, the threshold decreases by one (from t to $t-1$) for a limited escalation period. This models the real-world pattern of "if two of three aren't available, one can approve with a time delay."
4. **Delegation.** An unavailable approver delegates their approval capability to a substitute for a bounded time period.

Proposition 1 (Escalation Security Bound)**.** *Under the threshold reduction escalation policy, the minimum effective threshold during escalation is* $\max(1, t - e_{\max})$ *where* $e_{\max}$ *is the governance-set maximum escalation depth. If* $e_{\max} < t - 1$*, then escalation cannot reduce the threshold to single-signer authorization, preserving the multi-party security guarantee even during escalation.*

4 Social Recovery

Seed phrases are the original sin of cryptocurrency UX. A 24-word mnemonic is a single point of failure that users are told to write on paper and store in a safe. This is not a product experience. It is an abdication of design responsibility.

Social recovery replaces seed phrases with threshold key recovery through trusted contacts (guardians).

4.1 Guardian Model

Definition 3 (Guardian Set)**.** *A guardian set* $\mathcal{G} = (t, n, \{G_1, \ldots, G_n\})$ *is a set of* n *contacts, any* t *of whom can cooperate to recover the user's capsule access key. Each guardian* G_i *holds an encrypted key share* s_i *stored in their own capsule.*

The guardian selection UX presents this as: "Choose 5 trusted contacts. If you ever lose access to your account, any 3 of them can help you recover it." The user selects contacts from their address book. The system:

1. Generates a (t, n) Shamir sharing of the capsule recovery key.
2. Encrypts each share s_i under guardian G_i's public key.
3. Stores each encrypted share in G_i's capsule as a "recovery share" object (the guardian is notified: "[User] has named you as a recovery contact").
4. Records the guardian set commitment (hash of the guardian DIDs and threshold) on the I-Chain.

4.2 Recovery Protocol

When a user needs to recover (lost device, forgotten passphrase), the recovery protocol proceeds:

1. User authenticates via an alternate channel (email, phone, in-person verification with a guardian).
2. User requests recovery through the Lux client, which contacts all n guardians.
3. Each guardian receives: "[User] is requesting account recovery. Do you want to help? [Confirm] [Deny]." The guardian may additionally verify the request out-of-band (phone call, video chat).
4. When a guardian confirms, they authorize their capsule to release the encrypted share to the T-Chain recovery coordinator.
5. When t shares are collected, the T-Chain coordinates threshold reconstruction via a secure MPC protocol that reveals the recovery key only to the user's new device.
6. The user's capsule is re-keyed: a new primary key is generated, the capsule is re-encrypted, and the guardian shares are refreshed (proactive secret sharing [7]).

Theorem 2 (Recovery Security)**.** *Under the (t, n) guardian model, an adversary must compromise at least t guardians to recover a user's capsule key. If guardians are selected from independent social circles (family, work, friends), the probability of t simultaneous compromises is:*

$$\Pr[\textit{adversarial recovery}] \leq \binom{n}{t} \cdot p^t \tag{2}$$

where p is the per-guardian compromise probability. For $t = 3$, $n = 5$, and $p = 0.01$, this yields $\Pr \leq 10^{-5}$.

4.3 Guardian Rotation

Guardians change over time: friends drift apart, employees leave companies, contacts change devices. The guardian rotation protocol allows updating the guardian set without re-sharing the original key:

1. User initiates guardian rotation from the settings UI.
2. The system performs proactive secret sharing: generating new shares for the updated guardian set without reconstructing the secret.
3. Old shares held by removed guardians become cryptographically useless (they correspond to a deprecated polynomial).
4. New guardians receive their shares; the I-Chain commitment is updated.

This uses Herzberg et al.'s proactive secret sharing protocol [7], which refreshes shares without reconstructing the underlying secret—the recovery key is never materialized during rotation.

5 Shared Workspaces

A shared workspace is a capsule whose access control policy grants multiple users read, write, or admin permissions. The threshold primitive enables fine-grained collaborative access without a central server that holds all keys.

5.1 Access Control via Threshold Decryption

Workspace data is encrypted under a workspace key k_W. The key is not held by any single user. Instead, it is distributed as a (t, n) threshold key among workspace members:

Definition 4 (Workspace Threshold Access)**.** *A workspace W with members $\{M_1, \ldots, M_n\}$ and access threshold t encrypts data under key k_W where k_W is a threshold key requiring t member key shares for decryption. Different access levels map to different thresholds:*

- ***Read**: $t_{read} = 1$ (any member can read)*
- ***Write**: $t_{write} = 1$ (any member can write, with CRDT merge for conflicts)*
- ***Admin**: $t_{admin} = \lceil n/2 \rceil + 1$ (majority required for policy changes)*

For the common case of $t_{\text{read}} = 1$, each member holds a copy of the workspace key encrypted under their personal key. This avoids the latency of threshold decryption for reads while preserving threshold control for administrative actions.

5.2 Collaborative Editing with CRDTs

Workspace documents use Conflict-free Replicated Data Types (CRDTs) [8] for concurrent editing. Each member edits their local encrypted copy, generating CRDT operations that are encrypted and broadcast to other members via relay providers. Merging happens locally on each member's device:

$$\mathsf{Doc}_{M_i} = \mathsf{Merge}(\mathsf{Doc}_{M_i}, \mathsf{Decrypt}(k_W, \mathsf{ops}_{M_j})) \tag{3}$$

No server sees plaintext. The CRDT guarantees eventual consistency: all members converge to the same document state regardless of the order in which operations are applied.

5.3 Member Management

Adding or removing workspace members requires threshold admin approval:

1. A member proposes adding/removing a user.
2. The proposal is submitted as a T-Chain threshold approval flow with t_{admin} threshold.
3. When approved, the workspace key is re-shared to include/exclude the affected member.
4. For member removal, the workspace key is rotated (generating a fresh key and re-encrypting all data) to ensure the removed member cannot decrypt future updates.

Proposition 3 (Forward Secrecy on Member Removal)**.** *After a member M_r is removed and the workspace key is rotated, M_r cannot decrypt any data written after the rotation epoch, even if M_r retains their old key share. This follows from the key independence property: the new workspace key k'_W is generated independently of k_W and is never encrypted under M_r's personal key.*

6 Treasury Management

Multi-party spend authorization is the most natural application of threshold signatures. The product framing: "Payments above $X require approval from Y of Z signers."

6.1 Tiered Spend Policies

Definition 5 (Tiered Spend Policy)**.** *A tiered spend policy* $\mathcal{T} = \{(\ell_1, t_1), (\ell_2, t_2), \ldots, (\ell_k, t_k)\}$ *defines spending tiers where* ℓ_i *is the maximum transaction amount for tier* i *and* t_i *is the required approval threshold. Tiers are ordered:* $\ell_1 < \ell_2 < \cdots < \ell_k$ *and* $t_1 \leq t_2 \leq \cdots \leq t_k$*.*

A typical configuration for a startup:

Table 1: Example tiered spend policy for a three-person founding team.

Tier	**Amount**	**Approval**
Petty cash	$\leq$ \$500	1 of 3 (any founder)
Operating expense	$\leq$ \$10,000	2 of 3
Capital expenditure	$\leq$ \$100,000	3 of 3
Extraordinary	$>$ \$100,000	3 of 3 + 24h time lock

The user sees a payment approval screen: "This payment of \$15,000 requires approval from 2 of 3 signers. Alice has approved. Waiting for Bob or Carol." They do not see FROST partial signatures or Lagrange interpolation.

6.2 Time Locks and Velocity Limits

High-value actions include additional safeguards:

1. **Time lock.** After t approvals are collected, execution is delayed by Δt_{lock} (e.g., 24 hours). During this window, any signer can veto, canceling the transaction. This prevents compromised signers from executing irreversible transactions before the compromise is detected.

2. **Velocity limit.** Total spend within a rolling window W is capped at $V_{\max}$:

$$\sum_{tx \in \text{window}(t,W)} \mathsf{amount}(tx) \leq V_{\max} \tag{4}$$

   If the limit is reached, all further transactions require the highest approval tier regardless of individual transaction size.

Proposition 4 (Time Lock Defense Against Key Compromise)**.** *If an adversary compromises* t *signer keys at time* t_0*, the time lock ensures that no unauthorized transaction executes before* $t_0 + \Delta t_{lock}$*. If any honest signer detects the compromise within* Δt_{lock} *and exercises their veto, the adversary gains nothing. The expected loss under key compromise is therefore:*

$$\mathbb{E}[loss] = V_{\max} \cdot \Pr[\textit{no honest signer detects within } \Delta t_{lock}] \tag{5}$$

For $\Delta t_{lock} = 24h$ *and* $n - t \geq 1$ *honest signers with continuous monitoring,* $\Pr[\textit{no detection}] \to 0$*.*

7 Regulated Access

Certain data—personal health information, financial records, legal documents—is subject to regulatory requirements that mandate specific access controls. Threshold cryptography enables compliance without centralized key escrow.

7.1 Compliance-Gated Reveal

Definition 6 (Compliance Gate)**.** *A compliance gate $\mathcal{C} = (\pi, t_c, \{C_1, \ldots, C_{n_c}\})$ requires that a data reveal operation satisfies policy π (evaluated via TFHE on the T-Chain) and is co-signed by t_c of n_c compliance officers. The reveal is a threshold decryption operation:*

$$\mathsf{Reveal}(d) = \begin{cases} \mathsf{ThreshDec}(t + t_c, \{s_i\}_{i \in S \cup C}) & \text{if } \mathsf{Eval}(\pi) = 1 \\ \perp & \text{otherwise} \end{cases} \tag{6}$$

where S is the set of user-authorized shares and C is the set of compliance officer shares.

The UX presents this as: "Viewing this record requires compliance approval. A compliance officer has been notified." The compliance officer sees: "[User] requests access to [record type]. Reason: [stated reason]. [Approve] [Deny]." No cryptographic ceremony is visible to either party.

7.2 Audit Trail

Every compliance-gated reveal produces an on-chain receipt recording:

- Who requested the reveal (user DID)
- Who approved (compliance officer DIDs)
- When the reveal occurred (block timestamp)
- What was revealed (hash of the decrypted data, not the data itself)
- Why (the policy π that was satisfied)

This provides a tamper-evident audit trail that satisfies regulatory requirements (HIPAA access logs, GDPR data access records, SOC 2 access controls) without requiring a centralized audit service.

8 Progressive Security

New users should not face threshold setup on day one. A notes app with three entries does not need 2-of-3 guardian recovery. A wallet with $10 does not need multi-party spend approval. Progressive security matches the security model to the value at risk.

8.1 Security Tiers

Definition 7 (Progressive Security Model)**.** *A progressive security model $\mathcal{S} = \{(\nu_1, \Phi_1), (\nu_2, \Phi_2), \ldots\}$ maps value thresholds ν_i to security configurations Φ_i. When the value protected by a capsule crosses threshold ν_i, the system prompts the user to upgrade to configuration Φ_i.*

Typical progression:

Table 2: Progressive security tiers with automatic upgrade prompts.

Tier	Trigger	Auth	Recovery
Basic	Account creation	Device key only	Email recovery
Standard	$>$ 100 records or \$100	Device + passphrase	2-of-3 guardians
Enhanced	$>$ 1000 records or \$10,000	Device + biometric	3-of-5 guardians
Maximum	$>$ \$100,000	Hardware key + biometric	3-of-5 + time lock

8.2 Upgrade Protocol

When a user's capsule crosses a value threshold, the upgrade prompt appears: "Your account now holds significant value. We recommend adding recovery contacts for additional security. [Set Up Now] [Remind Me Later]."

If the user accepts:

1. The system generates a new (t, n) sharing of the capsule key appropriate for the new tier.
2. The user selects guardians (or the system suggests contacts based on communication frequency, with user confirmation).
3. Guardian shares are distributed as described in Section 4.
4. The old security configuration is replaced. If the previous tier had no threshold (1-of-1), this is the first introduction of threshold cryptography—but the user experiences it as "adding recovery contacts," not "distributing Shamir shares."

Remark 1. *The "Remind Me Later" option is genuine: users are never forced to upgrade. However, the system records the refusal and surfaces the prompt again after a reasonable interval (e.g., 30 days) or after the next significant value increase. The balance between security nudging and user autonomy is a product design decision, not a cryptographic one.*

8.3 Downgrade Prevention

Once a user upgrades to a threshold security tier, downgrading to a lower tier requires approval from the current threshold. A user cannot unilaterally remove their guardians, because the entire point of threshold security is that no single party (including the user alone at the enhanced tier) can make unilateral security decisions.

Proposition 5 (Downgrade Resistance)**.** *Under the progressive security model, downgrading from tier Φ_i with threshold t_i to tier Φ_j with threshold $t_j < t_i$ requires t_i approvals. An adversary who compromises the user's device (one share) cannot downgrade to single-signer mode without $t_i - 1$ additional compromises.*

9 UX Interaction Patterns

The following patterns define how threshold operations are presented to users across all the product experiences described above.

9.1 Notification-Driven Approval

Threshold operations are initiated by one party and completed by others via push notifications. The initiator sees a pending state ("Waiting for 2 more approvals"). Approvers see an actionable notification ("[User] requests your approval for [action]. [Approve] [Deny]").

Design rules:

1. Notifications must describe the action in plain language, never in cryptographic terms.
2. The approval action is a single tap/click, not a multi-step process.
3. The approver can inspect details (amount, recipient, document name) before approving.
4. Rejection is always available and requires no justification (though the system may prompt for one).

9.2 Progress Indicators

Users waiting for threshold completion see a progress indicator: "2 of 3 approvals received. Waiting for Alice or Bob." The indicator updates in real time as approvals arrive. This maps the abstract concept of "quorum assembly" to the familiar concept of "waiting for responses."

9.3 Delegation

When an approver is unavailable, they can delegate their approval capability to a substitute for a bounded time period. The UX presents this as: "You have a pending approval request but marked yourself as away. Would you like to delegate to [substitute]? They will be able to approve on your behalf until [date]."

Delegation is implemented as a threshold key share re-encryption: the approver's share s_i is re-encrypted under the delegate's public key via a proxy re-encryption scheme [9], without revealing s_i to any intermediary. The delegation expires automatically (the re-encryption key has a time-bounded validity enforced by the T-Chain).

9.4 Recovery UX

The recovery experience avoids all cryptographic terminology:

1. User on new device: "I lost access to my account."
2. System: "We'll contact your recovery contacts. 3 of 5 need to confirm it's you."
3. Each guardian: "[User] says they lost access. Can you confirm this is really them? You may want to call them to verify. [Confirm] [This Isn't Them]"
4. When 3 confirm: "Your account has been restored to this device."

The user never learns what Shamir secret sharing is. They experience a social verification process that happens to be cryptographically enforced.

10 Security Analysis

10.1 Threshold Security

Theorem 6 (Threshold UX Security Equivalence)**.** *The security of each Threshold UX pattern (approval flows, social recovery, shared workspaces, treasury management, regulated access) is equivalent to the security of the underlying threshold cryptographic primitive (FROST, Shamir sharing, threshold decryption), under the assumption that the UX layer is a faithful mapping (it does not weaken the threshold, bypass quorum requirements, or leak key material through side channels).*

Proof. Each UX pattern maps one-to-one to a threshold operation:

- Approval tap $\leftrightarrow$ FROST partial signature
- Guardian confirmation $\leftrightarrow$ Shamir share release
- Workspace admin action $\leftrightarrow$ threshold decryption
- Spend authorization $\leftrightarrow$ threshold signature
- Compliance approval $\leftrightarrow$ threshold co-decryption

The UX layer transforms user input into the corresponding cryptographic operation without modification. No additional attack surface is introduced beyond what exists in the underlying protocol. The mapping is injective: each user action produces exactly one cryptographic operation, and no cryptographic operation can be triggered without the corresponding user action. □

10.2 Social Engineering Resistance

The primary new attack vector in Threshold UX is social engineering: an attacker convinces t guardians to approve a malicious recovery or transaction. Mitigations:

1. **Out-of-band verification prompts.** The system suggests (but cannot enforce) that guardians verify recovery requests via a different channel (phone call, in-person meeting).
2. **Time delays.** Recovery operations include a mandatory waiting period (e.g., 48 hours) during which the legitimate user can cancel the recovery if they still have access.
3. **Guardian diversity.** The system encourages selecting guardians from independent social circles (family, work, friends) to reduce the probability that an attacker has social access to t guardians simultaneously.
4. **Anomaly detection.** The T-Chain monitors recovery patterns and flags anomalies (e.g., recovery requested from a new geographic region, multiple recovery attempts in a short period).

Proposition 7 (Social Engineering Bound)**.** *If guardians are drawn from k independent social circles with at most $\lfloor n/k \rfloor$ guardians per circle, and the threshold $t > \lfloor n/k \rfloor$, then an attacker who infiltrates a single social circle cannot achieve the threshold. Full social engineering requires simultaneous infiltration of at least $\lceil t \cdot k/n \rceil$ independent circles.*

11 Conclusion

Threshold cryptography has been ready for consumer deployment for years. What has been missing is the product design that makes it invisible. Threshold UX closes this gap by mapping threshold primitives to product patterns that users already understand: approval flows, recovery contacts, shared workspaces, spending limits, and compliance gates.

The key contributions of this paper are:

1. **Threshold as UX primitive.** A formal mapping from threshold cryptographic operations (FROST signatures, Shamir sharing, threshold decryption, TFHE policy evaluation) to product interaction patterns (approvals, recovery, workspaces, treasury, compliance).
2. **Social recovery without seed phrases.** A guardian-based recovery protocol that replaces 24-word mnemonics with "3 of 5 trusted contacts," including guardian rotation via proactive secret sharing.
3. **Progressive security.** A tiered security model that starts at 1-of-1 and upgrades to threshold control as value increases, with downgrade resistance preventing unilateral weakening.
4. **UX interaction patterns.** Concrete design patterns (notification-driven approval, progress indicators, delegation, timeout escalation) that hide cryptographic ceremony behind familiar product metaphors.

5. **Security equivalence proof.** A demonstration that the UX layer preserves the security guarantees of the underlying threshold primitives, introducing no additional attack surface beyond social engineering, which is bounded by guardian diversity.

Threshold cryptography is not hard to use. It has been hard to present. That problem is now solved.

References

[1] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[2] R. Gennaro and S. Goldfeder, "Fast multiparty threshold ECDSA with fast trustless setup," in *ACM CCS*, 2018.

[3] C. Komlo and I. Goldberg, "FROST: Flexible round-optimized Schnorr threshold signatures," in *SAC*, Springer, 2020.

[4] R. Canetti, R. Gennaro, S. Goldfeder, N. Makriyannis, and U. Peled, "UC non-interactive, proactive, threshold ECDSA with identifiable aborts," in *ACM CCS*, 2020.

[5] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *STOC*, ACM, 2009.

[6] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, "TFHE: Fast fully homomorphic encryption over the torus," *Journal of Cryptology*, vol. 33, pp. 34–91, 2020.

[7] A. Herzberg, S. Jarecki, H. Krawczyk, and M. Yung, "Proactive secret sharing or: How to cope with perpetual leakage," in *CRYPTO*, Springer, 1995.

[8] M. Shapiro, N. Preguiça, C. Baquero, and M. Zawirski, "Conflict-free replicated data types," in *SSS*, Springer, 2011.

[9] M. Blaze, G. Bleumer, and M. Strauss, "Divertible protocols and atomic proxy cryptography," in *EUROCRYPT*, Springer, 1998.

[10] J. Bonneau, C. Herley, P. C. van Oorschot, and F. Stajano, "The quest to replace passwords," in *IEEE S&P*, 2012.

[11] S. Eskandari, D. Barrera, E. Stobert, and J. Clark, "A first look at the usability of Bitcoin key management," in *NDSS Workshop on Usable Security*, 2018.

[12] V. Shoup, "Practical threshold signatures," in *EUROCRYPT*, Springer, 2000.