Interactive Multi-Credential Authentication

Deepak Maram
Cornell Tech and Mysten Labs

Mahimna Kelkar
Cornell Tech and IC3

Ittay Eyal
Technion and IC3

ABSTRACT

Authentication is the first, crucial step in securing digital assets like cryptocurrencies and online services like banking. It relies on principals maintaining exclusive access to credentials like cryptographic signing keys, passwords, and physical devices. But both individuals and organizations struggle to manage their credentials, resulting in loss of assets and identity theft.

In this work, we study mechanisms with back-and-forth *interaction* with the principals. For example, a user receives an email notification about sending money from her bank account and is given a period of time to abort.

We define *the authentication problem*, where a *mechanism* interacts with a user and an attacker. A mechanism's success depends on the scenario—which credentials each principal knows. The *profile* of a mechanism is the set of scenarios in which it succeeds. The subset relation on profiles defines a partial order on mechanisms. We bound the profile size and discover three types of novel mechanisms that are *maximally secure*.

We show the efficacy of our model by analyzing existing mechanisms and make concrete improvement proposals: Using "sticky" messages for security notifications, prioritizing credentials when accessing one's bank account, and using one of our maximal mechanisms to improve a popular cryptocurrency wallet. We demonstrate the practicality of our mechanisms by implementing the latter.

ACM Reference Format:
Deepak Maram, Mahimna Kelkar, and Ittay Eyal. 2024. Interactive Multi-Credential Authentication. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 22 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Authentication plays a critical role in safeguarding online services and digital assets. An *authentication mechanism* binds a user's physical identity to a digital identity [11, 16]. It relies on the user's exclusive access to *credentials* like a password, a one-time PIN (OTP), or a cryptographic signing key. But credential management is challenging [20, 24, 37]: identity theft in the US is rampant [26]; an estimated 20% of Bitcoin have disappeared as a result of key loss [23]; and $600M in cryptocurrency were stolen recently due to a single company's key mismanagement [33].

Some practical mechanisms employ *interaction*. For example, the Argent cryptocurrency wallet [6] uses multiple cryptographic signing keys as credentials. In a possible configuration, any 1-out-of-3 credentials can initiate a change of credentials, but the change only happens after two days, during which the operation can be cancelled with any 2-out-of-3 credentials. In online banking, some banks (e.g., [3]) introduce an artificial delay period before transferring funds; they notify the client and allow her to abort an erroneous transaction during this period. Government agencies (e.g., [31]) send a notice by (physical) mail about reversible account activity, allowing victims to revert in case of identity theft attempts [26].

But despite being used in practice, to the best of our knowledge, interactive authentication has never been formally studied, and its advantages are therefore underutilized. Prior work (§2) focused on proposing frameworks that only model non-interactive mechanisms [25, 29], or consider specific interactive mechanisms like the ones mentioned above [6, 43], which we show to be sub-optimal. Multi-factor authentication is typically defined as requiring multiple credentials [41], i.e., combine credentials using a conjunction (AND) operator, without taking advantage of interactivity. Similarly, multi-sig or threshold mechanisms (k-out-of-n keys) are popular in the cryptocurrency industry [15, 27, 30], but are non-interactive.

In this work, we formally study interactive authentication mechanisms. We first define the *authentication problem* (§3), consisting of a *mechanism* and two players: a *user* and an *attacker* that stands for all entities trying to authenticate as the user. The mechanism is a *deterministic finite automaton* using $n \geq 1$ credentials to identify the user. Each credential can be in one of four states [25]: *safe* (only the user has it), *stolen* (only the attacker has it), *leaked* (both have it) or *lost* (neither has it). The states of all credentials define a *scenario*, with a total of 4^n scenarios possible.

The players send to the mechanism messages carrying proofs of the credentials they have, and eventually the mechanism *decides* which of them is the user. A mechanism *succeeds* in a scenario if it is correct irrespective of what the attacker does; otherwise it *fails*. When the communication channels are *synchronous*, the mechanism can rely on an interactive message exchange with the players.

To evaluate mechanisms, we define the *security profile* of a mechanism to be the set of all scenarios in which it is successful. For example, with two credentials denoted by c_1 and c_2, consider the OR-mechanism where either c_1 or c_2 can be used to authenticate, i.e., $c_1 \vee c_2$. Its profile has three scenarios: (c_1 and c_2 safe), (c_1 safe, c_2 lost) and (c_1 lost, c_2 safe). The mechanism succeeds in the latter two because the user knows enough credentials to authenticate and the attacker does not. Note that the mechanism fails in scenarios where both players have 1–2 credentials, e.g., (c_1 stolen, c_2 safe), because we conservatively assume that the adversary controls the order in which messages are delivered to the mechanism.

The security profile defines a relation between any two mechanisms M_1 and M_2 using the same number of credentials: M_1 is *better* than M_2 if the profile of M_1 is a superset of the profile of M_2. M_1 is *equivalent* to M_2 if they have the same profile. Otherwise, M_1 and M_2 are incomparable. A mechanism is *maximally secure* or simply *maximal* if no other mechanism is better. For example, if a

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2-credential mechanism M succeeds in all three scenarios where the OR-mechanism is successful plus some additional scenarios, then we can say that M is better than the 2-credential OR-mechanism.

By proving constraints on the profile sets any mechanism could achieve, we bound the profile size (§4). We find that with n-credentials, no mechanism succeeds in more than $P(n) = \frac{4^n - 2^n}{2}$ scenarios. For example, $P(2) = 6$, that is, a 2-credential mechanism can win in at most 6 scenarios. Using automatons as a generic computational model enables us to prove this impossibility result.

The bound is tight as we discover several maximal mechanisms meeting it (§5). We discover a generic class of maximal mechanisms that follow a simple template: Either player can initiate the mechanism, at which point the mechanism starts a timer. Until the time runs out, each player can submit messages to the mechanism, carrying one or more credentials. Finally, the mechanism decides on the winner. We refer to mechanisms (maximal or otherwise) following the above structure as *bounded-delay mechanisms*. They only differ in the way a winner is decided.

We identify three sufficient properties for a bounded-delay mechanism to be maximal: (1) *ID-Agnostic*: the mechanism only decides based on messages, not other information; (2) *Knowledge-rewarding*: if a player submits a credential set that is a superset of its counterpart's set, then the mechanism decides on the former; and (3) *Transitive-knowledge-rewarding*: submitting additional credentials cannot lead to a worse outcome for the submitting player.

We discover three classes of bounded-delay mechanisms that satisfy these properties. The first two are fairly intuitive: *Majority mechanisms* choose whoever submits the most credentials, with some tie-breaking function. *Priority mechanisms* use a priority vector defined over the n-credentials, and the winner is the player that submits a unique higher priority credential. For example, consider the priority vector $[c_2,c_3,c_1]$: if player P1 submits $\{c_1,c_2\}$ and player P2 submits $\{c_2,c_3\}$, P2 wins due to the higher priority of c_3 over c_1. The third class is a variation of priority mechanisms.

We can now illustrate the advantage of interactivity by going back to the 2-credential setting. Consider the 2-credential priority mechanism defined by the vector $[c_1,c_2]$. Its profile contains (c_1 and c_2 safe); (c_1 safe, c_2 leaked or lost or stolen) because only the user knows the highest-priority credential c_1; and (c_1 leaked or lost, c_2 safe) because c_2 is safe and c_1 is known to either both the players or to neither. In total, the size of the profile is six, equal to the upper bound $P(2)$ that any 2-credential mechanism can reach. Note that the priority mechanism offers better security than the OR-mechanism because the former's profile is a superset of the latter with three more scenarios: (c_1 safe, c_2 leaked or stolen) and (c_2 safe, c_1 leaked).

The three mechanism classes cover the *complete set* of n-credential maximal mechanisms for $n \leq 3$ (§6), i.e., *any* mechanism *must be either worse than or equivalent* to a mechanism in the three classes. It follows that for any given probability distribution of credential states, one of the complete set mechanisms is the most secure, i.e., has the highest probability of being secure.

We show the efficacy of our model by analyzing deployed interactive mechanisms (§7). Specifically, we examine the widely used cryptocurrency wallet Argent [6] used by over a million users [12] and a prominent bank [3] that serves over a 100 million users. We apply our framework to model their mechanisms, accurately assess their security levels, pinpoint existing vulnerabilities, and propose enhancements.

Argent lets the user select m key guardians (e.g., friends) and keeps one key on the user's phone for a total of $n = m+1$ credentials. It provides functionality to transfer funds and modify credentials. We find that Argent's profile has 5 scenarios if $n = 2$ and 22 scenarios if $n = 3$ when used according to the Argent documentation. Therefore, Argent achieves much better security than commonly used non-interactive designs, e.g., the popular 1-out-of-2 and 2-out-of-3 wallets only succeed in 3 and 14 scenarios respectively.

However, our analysis also reveals that Argent is not maximally secure, as it succeeds in 6 scenarios when $n = 2$ and in 28 scenarios when $n = 3$. This is attributable to three reasons: (i) the wallet's inability to execute multiple transactions atomically, (ii) the absence of a prioritization scheme among guardians and (iii) the implementation of a feature termed "fast withdrawals", that facilitates instant money transfers without delays. Argent's security can be improved either by fixing these issues individually or by adopting our maximal mechanisms, which has the added benefit of vastly simplifying the design.

We illustrate the practicality of our mechanisms through the implementation of a priority mechanism in Solidity (App. A). Our implementation incurs fees similar to those of Argent (§7.1).

Similarly, we analyze a (slightly) simplified version of the authentication mechanism employed for logging into the website for HDFC bank—the largest bank in India (§7.2). We find that the mechanism's security is over-reliant on the security of a one-time password (OTP), received on a registered mobile device. For example, the bank's protocol permits an instantaneous password reset with an OTP. Using a 2-credential priority mechanism before a password reset instead can significantly enhance HDFC's security. Specifically, this simple change can improve the security profile from 20 to 28 scenarios, thus achieving maximal security.

Lastly, a practical insight arising from this work is the criticality of the assumed synchronous channel. In practice, such a channel must be implemented. For example, consider email (or text message) serving as the notification channel from a bank to its clients. If the attacker gained even temporary access to a user's mobile device, she could delete any notifications, thus voiding the channel's effectiveness. We propose to overcome this vulnerability by making such notification emails *sticky* – they cannot be deleted for, say, 24 hours. In other settings such as blockchain smart contracts, the notifications are public, but the user should use multiple devices to monitor the chain [1] (as used for other goals, e.g., [28, 36, 38]).

This work opens the door to a plethora of questions on authentication mechanisms in alternative models (§8), e.g., with asynchronous communication channels, under different knowledge models and with usability considerations. Addressing those is the next step towards secure authentication mechanisms fit for contemporary challenges.

In summary (§9), our main contributions are:

(1) A definition of the authentication problem, in particular under synchrony,
(2) a metric of the security level of an authentication mechanism, namely security profiles,
(3) a tight upper bound on the profile size of any mechanism,

(4) novel maximally-secure n-credential mechanisms,
(5) the complete sets of maximal mechanisms for $n \leq 3$,
(6) security analysis of deployed mechanisms with concrete improvement proposals, and
(7) Solidity implementation of a maximally-secure wallet.

2 RELATED WORK

To the best of our knowledge, previous work did not provide a general definition of the authentication problem or analyzed the interactive authentication design space.

The closest work is Eyal's cryptocurrency wallet design [25], which investigates what the best mechanism is, but in a restrictive setting with only boolean formulae, i.e., one-shot mechanisms. Our maximal mechanisms have larger security profiles. Hammann et al. [29] use a similarly restrictive model and their focus is orthogonal to ours, namely, to uncover vulnerabilities arising from the links between different mechanisms a user uses. Mouallem and Eyal [40] explore the authentication problem but in asynchronous settings; mechanisms there are substantially weaker, underscoring the advantage of using interactive mechanisms when possible.

There is substantial interest in analyzing the security of multi-factor authentication mechanisms. For example, Jacomme et al. [32] and Barbosa et al. [14] analyze Google's 2FA and Fido's U2F. But they do not propose generic frameworks; moreover, these mechanisms are also one-shot.

Elaborate authentication mechanisms are common for cryptocurrency assets [10, 15, 21, 30]; some of which even use interactive schemes like Argent [6], SmartCustody [8] and CoinVault [13] (which was part of the inspiration for this work). Interactive designs are sometimes seen in other contexts too, e.g., e-commerce platforms often provide customers with a limited timeframe to cancel orders. Some financial institutions and governmental agencies adopt it for enhanced account security [3, 31]. These demonstrate the practicality of interactive schemes, but also the need for rigor, as we demonstrate by analyzing and proposing improvements for the popular Argent cryptocurrency wallet and for HDFC bank access in §7.

Paralysis Proofs [43] leverages interactivity to deal with credential loss; similar ideas were recently proposed in the Bitcoin community [8]. We discuss this approach (§7.3) and show that our mechanisms have larger security profiles.

Vaults [39] (the first interactive cryptocurrency wallet design we know of) and KELP [17] leverage interactivity to deal with key leakage and loss, respectively. But their models are distinct from ours: Vaults [39] treats the case where neither player can withdraw funds as a success, since it removes the motivation for an attack; we conservatively treat this outcome as a failure. KELP [17] considers temporary exclusive knowledge to the user – a user knows about a lost key before anyone else; in §8 we consider a stronger model where the user can't tell whether a key is lost or stolen. Both works consider specific mechanisms while we find bounds and maximal mechanisms.

Authorization research has focused on implementing security policies [18, 19, 42]. In some contexts, our mechanisms can be implemented using those solutions.

3 MODEL

We formalize the authentication problem. We describe an execution, its participants and communication (§3.1), credentials (§3.2), the automaton model for authentication mechanisms (§3.3), and player strategies (§3.4), all summarized in Algorithm 1. Finally, we define security profiles, with which we evaluate mechanisms (§3.5).

3.1 Execution: participants, time and network

The system comprises an *authentication mechanism* M and two players, a *user* U and an *attacker* A.

An execution begins with the environment (an entity we use to orchestrate a real-world situation) assigning distinct *player identifiers* to the user and to the attacker from the set $\mathcal{P} = \{0,1\}$. It picks $\gamma \in \mathcal{P}$ and assigns γ^{U} to the user and γ^{A} to the attacker, e.g., if $\gamma = 0$, then the user is player 0 and the attacker is player 1. The identifiers are akin to cookies used to identify a website visitor during a single session. We refer to γ as the *environment's strategy*.

Time progresses in discrete steps. During the execution, both parties interact with the mechanism by sending and receiving messages. Communication channels are reliable and synchronous – if a message is sent in time step a, it reaches the recipient in step a+1.

The attacker controls message order within a time step. That is, if both user and attacker send messages to the mechanism at the same time step, the attacker can choose which of the two is received first.

The mechanism decides either 0 or 1 to end an execution, and the player with this identifier *wins*. This is a one-time irrevocable operation corresponding to, say, withdrawing money out of a bank.

3.2 Credentials, scenarios and messages

The system contains a set of n credentials, $C_{\mathsf{all}} = \{c_1, c_2, \ldots, c_n\}$. Each credential is in one of four states [25]: (1) *Safe:* Only U has it, (2) *Lost:* No one has it, (3) *Leaked:* Both U and A have it, or (4) *Stolen:* Only A has it. Denote the state of credential c_i by σ_i, so for all $1 \leq i \leq n : \sigma_i \in \{\mathsf{safe}, \mathsf{lost}, \mathsf{leaked}, \mathsf{stolen}\}$. A *scenario* σ is a vector of the n credential states and $\Sigma = \{\mathsf{safe}, \mathsf{lost}, \mathsf{leaked}, \mathsf{stolen}\}^n$ is the set of all scenarios, i.e. $\sigma \in \Sigma$.

The *credential set* of the user (resp., attacker) contains all the credentials available to that player, denoted $C^U_\sigma = \{c_i | \sigma_i \in \{\mathsf{safe}, \mathsf{leaked}\}\}$ (resp., $C^A_\sigma = \{c_i | \sigma_i \in \{\mathsf{leaked}, \mathsf{stolen}\}\}$).

At the start of an execution, the environment picks a scenario $\sigma \in \Sigma$ and initializes U and A with the credential sets C^U_σ and C^A_σ, respectively. The scenario σ is *common knowledge*, i.e., the state of all n credentials is known to both parties.

Each message sent by U or A consists of a player identifier $p \in \mathcal{P}$ and a set of credentials C. A message can only carry a subset of the credentials available to the sender, i.e., a user (resp., attacker) message can carry a set of credentials C s.t. $C \subseteq C^U_\sigma$ (resp., $C \subseteq C^A_\sigma$). To avoid credential leakage by listening to the channel, a message should actually contain encrypted credentials or zero knowledge proofs of credential knowledge, e.g., signatures for private keys etc. We avoid this detail for simplicity.

3.3 Authentication mechanism

The mechanism M is a deterministic one-clock timed automaton [7] known to both the user and attacker [34, 35], defined by the following elements:

States: The automaton has a finite set $\mathcal{T}$ of *states*. It includes a special *starting state* I_{init} and two disjoint sets of *final states*, $\mathcal{T}_0^{\text{fin}}$ and $\mathcal{T}_1^{\text{fin}}$. The execution ends as soon as the automaton reaches a final state, i.e., if $s \in \mathcal{T}_i^{\text{fin}}$, then the winner is the player with identifier i.

Clock: A *clock* has some value $t \in \mathbb{Z}_{\geq 0}$, initialized with $t=0$, and incremented by 1 in each step. The automaton can reset the clock back to its initial state ($t=0$).

A tuple (s,v) of an automaton state s and a clock state t is called the *system state*. At the start of an execution, we have $s \leftarrow I_{\text{init}}$ and $t \leftarrow 0$.

Transitions: An automaton transition is a 6-tuple consisting of a source and a destination state, three types of guards and a clock reset bit, as follows.

The *Clock Reset* $\mathcal{R} = \{\mathsf{True}, \mathsf{False}\}$ represents the reset of the clock state or no reset. Resetting means that the clock is set back to its initial state ($t \leftarrow 0$) upon taking the transition.

Transitions take place on message arrival. A guard g is a constraint specifying that the message must meet a condition for the transition to take place. A guard with the value $\perp$ indicates no constraint. There are three types of guards:

A *Player guard* means the message must be sent by a certain player ID. The set of player identifier guards is $\mathcal{G}_{\text{id}} = \mathcal{P} \cup \{\perp\}$. If a player $p \in \mathcal{P}$ satisfies the guard g^{plr} we denote $p \vdash g^{plr}$. For example, $0 \vdash g^{plr}$ only if $g^{plr} \in \{0, \perp\}$.

A *Credential guard* means that the message must carry certain credentials. The set of credential guards $\mathcal{G}_c$ includes all non-constant monotone boolean formulae of the availability of the credentials in the set C_{all} and $\perp$. For example, if $n=2$, then $\mathcal{G}_c = \{c_1, c_2, c_1 \wedge c_2, c_1 \vee c_2, \perp\}$. A credential set C satisfies a credential guard g^{cd}, denoted $C \vdash g^{cd}$, if the credentials in C satisfy the boolean formula g^{cd}. For example, if $g^{cd} = c_1 \vee c_2$ and $C = \{c_1\}$, then $C \vdash g^{cd}$.

A *Clock guard* is a condition on the clock state t expressed through a comparison operator $\sim \in \{<, \leq, =, \geq, >\}$ and a natural number $l \in \mathbb{N}$, i.e., $(t \sim l)$. The set of all clock guards (including $\perp$) is $\mathcal{G}_t$. For example, a guard $g^{clk} = (t < 5)$ specifies that the clock state t must be less than 5. Denote by $t \vdash g^{clk}$ that the clock state t satisfies the guard g^{clk}.

In summary, $\mathcal{D} \subseteq \mathcal{T} \times \mathcal{G}_{\text{id}} \times \mathcal{G}_c \times \mathcal{G}_t \times \mathcal{R} \times \mathcal{T}$ is the set of transitions of M. (The source state must be a non-final state, and the target state not the beginning state.) And an automaton M is a 6-tuple given by $M = (C_{\text{all}}, \mathcal{T}, clock, \mathcal{D}, \mathcal{T}_0^{\text{fin}}, \mathcal{T}_1^{\text{fin}})$.

If the current state is s and the automaton receives a message satisfying all the guards of an outgoing transition of s, then the destination state of the transition becomes the new state. There can exist at most one such transition because we consider a deterministic automaton, i.e., we require: For all states $s \in \mathcal{T}$, all clock states $t \in \mathbb{Z}_{\geq 0}$, all sets of credentials $C \subseteq C_{\text{all}}$, and all player identifiers $p \in \mathcal{P}$, there exists at most one transition $(s, g^{plr}, g^{cd}, g^{clk}, r, s') \in \mathcal{D}$ such that all the guards are satisfied, i.e., $p \vdash g^{plr}$, $C \vdash g^{cd}$ and $t \vdash g^{clk}$.

The automaton informs both parties of each state change, even if multiple transitions occur within a single time step.

Note 1. *We use a clock in automatons for illustrative reasons. It can be removed to construct a DFA using standard techniques (see region automata [7]).*

Algorithm 1 Execution with players U and A and mechanism $M = (C_{\text{all}}, \mathcal{T}, clock, \mathcal{D}, \mathcal{T}_0^{\text{fin}}, \mathcal{T}_1^{\text{fin}})$

Init: The environment chooses a scenario $\sigma \in \Sigma$ and $\gamma \in \{0,1\}$. It initializes U (resp., A) with the player identifier $\gamma^{\text{U}} = \gamma$ (resp., $\gamma^{\text{A}} = 1-\gamma$) and the set of credentials C_σ^U (resp., C_σ^A).

$s \leftarrow I_{\text{init}}$, $t \leftarrow 0$ ▷ State and clock

Execution:

repeat ▷ Execute loop once per time step
  $Y^{\text{U}} \leftarrow \{(\gamma^{\text{U}}, C) | C \in S^U((s,t))\}$ ▷ U's messages
  $Y^{\text{all}} \leftarrow S^A((s,t), Y^{\text{U}})$ ▷ A adds messages and orders them
  for $i = 1, 2, \ldots, |Y^{\text{all}}|$ **do** ▷ M processes messages
    if $\exists (s', g^{plr}, g^{cd}, g^{clk}, r, s'') \in \mathcal{D}$, $s = s'$,
      $p_i \vdash g^{plr}$, $C_i \vdash g^{cd}$, and $t \vdash g^{clk}$ **then**
      $s \leftarrow s''$ ▷ Update automaton state
      if $r = \mathsf{True}$ **then** $t \leftarrow -1$. ▷ Reset clock
      if $s \in \mathcal{T}_0^{\text{fin}}$ **then return** 0
      if $s \in \mathcal{T}_1^{\text{fin}}$ **then return** 1
  $t \leftarrow t+1$ ▷ Advance clock

3.4 Player strategies and mechanism success

The *user strategy* S^U specifies which messages the user sends in any given extended mechanism state:

Definition 1 (User strategy). *A user strategy is a function of the system state $e = (s,t)$, $S^U(e)$ that returns $\perp$ or an ordered list of credential sets $\{C_i\}$.*

Denote Y^{U} as the set of user messages that the user sends when the system state is e, i.e., $Y^{\text{U}} \leftarrow \{(\gamma^{\text{U}}, C) | C \in S^U(e)\}$.

Definition 2 (Attacker strategy). *An attacker strategy is a function of the system state e and user's messages Y^U, $S^A(e, Y^U)$ that returns an ordered list of messages $Y^{\text{all}} = \{p_i, C_i\}$ containing both user and attacker messages such that no user message is lost, i.e., $\forall (\gamma^U, C) \in Y^U, (\gamma^U, C) \in Y^{\text{all}}$.*

A user (resp., attacker) strategy is *playable* in a scenario σ if for all possible systems states e and for all $C_i \in S^U(e) : C_i \subseteq C_\sigma^U$ (resp., for all user messages Y^{U}, $(\gamma^{\text{A}}, C_i) \in S^A(e, Y^{\text{U}}) : C_i \subseteq C_\sigma^A$).

A player adopting a strategy S means that if the current extended state is e, then the user sends its messages $Y^{\text{U}} \leftarrow \{(\gamma^{\text{U}}, C) | C \in S^U(e)\}$, and after seeing user's messages, the attacker sends the final set of messages $S^A(e, Y^{\text{U}})$.

An *execution* E is fully defined by an automaton M, a scenario σ, player strategies S^U and S^A, and the environment's strategy γ, i.e., $E = (M, \sigma, S^U, S^A, \gamma)$. The details of how an execution unfolds are specified in Algorithm 1. The user wins only if the mechanism chooses it:

Definition 3 (Execution winner). *Given an execution $E = (M, \sigma, S^U, S^A, \gamma)$, the* winner *of an execution is the player decided by the automaton M, or the attacker A if M never decides. We denote $Win^{exec}(E) \in \{U, A\}$ accordingly.*

An environment strategy change can affect the outcome of two executions that use the same player strategies.

A mechanism M *succeeds* in a scenario σ, denoted $Suc(M, \sigma)$, if the user wins irrespective of other's strategies:

Definition 4 (Success). *Given a mechanism M and a scenario σ, we say that the mechanism is* successful, *denoted $Suc(M,\sigma)$, if the user consistently wins against any attacker's strategy and environment's strategy, i.e., $\exists S^U_{\text{win}}: \forall S^A, \forall \gamma, E=(M,\sigma,S^U_{\text{win}},S^A,\gamma)$ and $Win^{exec}(E)=U$. Otherwise, the mechanism fails, denoted $\neg Suc(M,\sigma)$.*

Let $X=(M,\sigma)$ denote a mechanism and a scenario, and call X an *extended scenario*. We say $Win_X(S^U,S^A)=\text{U}$ if the strategy S^U wins for the user in all executions where the attacker employs S^A, i.e., independent of the environment's strategy. A strategy S^U_{win} is a winning user strategy if it wins against any attacker strategy, i.e., $\forall S^A: Win_X(S^U_{\text{win}},S^A)=\text{U}$.

On the other hand, $Win_X(S^U,S^A)=\text{A}$ means that S^A wins for the attacker in some execution where the user employs S^U, i.e., with some player identifier allocation. And S^A_{win} is a winning strategy for the attacker if $\forall S^U: Win_X(S^U,S^A_{\text{win}})=\text{A}$. Note that if the user doesn't have a winning strategy, the attacker does due to Zermelo's Theorem (App. D).

3.5 Mechanism profiles

Having defined mechanism success, we can now evaluate and compare mechanisms. A mechanism's *profile* is a concise representation of its security level containing all the scenarios in which it succeeds.

Definition 5 (Profile). *The profile of a mechanism M denoted by $prof(M)$ is the set of all scenarios where M succeeds, i.e., $prof(M)=\{\sigma \mid \sigma\in\Sigma \wedge Suc(M,\sigma)\}$.*

The profile can also be viewed as an n-dimensional matrix where n is the number of credentials. Each cell represents a distinct scenario and the value in it is 1 or 0 if the scenario is in the profile or isn't, respectively (e.g., Fig. 1).

The profiles define a partial order on the set of *all n-credential mechanisms*, denoted by $\mathcal{M}_n$. Two mechanisms $M,M'\in\mathcal{M}_n$ are *equivalent*, denoted by $M\cong M'$, if one's profile can be obtained from the other's by permuting the credential set:

Definition 6 (Equivalence). *Given a permutation $\pi : \{1,...,n\}\to\{1,...,n\}$ and a scenario σ, define the permuted scenario σ^π by $\sigma^\pi_i=\sigma_{\pi(i)}$. Then, $M\cong M'$ if $\exists\pi: \bigcup_{\sigma\in prof(M)}\sigma^\pi=prof(M')$.*

A mechanism M_1 is *better* than another mechanism M_2, denoted by $M_1\succ M_2$, or M_2 is *worse* than M_1, if $\exists M_3\cong M_2$ such that $prof(M_1)\supset prof(M_3)$. M_1 is *better than or equivalent to* M_2, denoted by $M_1\succeq M_2$, if $M_1\succ M_2$ or $M_1\cong M_2$ ($\preceq$ defined analogously). And M_1 is *incomparable* to M_2, denoted by $M_1\nsim M_2$, if $M_1\not\succeq M_2\wedge M_1\not\preceq M_2$.

We give an example to illustrate the relations. As we saw (§1), the profile of the 2-credential OR-mechanism $c_1\vee c_2$ has three scenarios: (c_1 and c_2 safe), (c_1 safe, c_2 lost) and (c_1 lost, c_2 safe). Now consider the AND-mechanism $c_1\wedge c_2$: its profile also has three scenarios, namely (c_1 and c_2 safe), (c_1 safe, c_2 leaked) and (c_1 leaked, c_2 safe). We can see that the two mechanisms are incomparable as neither is better than the other nor are they equivalent.

The relation naturally defines *maximal* mechanisms:

Definition 7 (Maximally secure mechanism). *A mechanism $M\in\mathcal{M}_n$ is* maximally secure *or* maximal *if for all $M'\in\mathcal{M}_n$: $M'\preceq M$ or $M'\nsim M$.*

4 PROFILE SIZE BOUND

We find an upper bound on the profile set size, i.e., given n, we find the maximum number of scenarios in which an n-credential mechanism succeeds (§4.2). Before that, we prove a few results useful in proving the bound (§4.1).

4.1 Useful results

First, we define *runs*. A *run* $r=\{(p_1,C_1,a_1),...,(p_z,C_z,a_z)\}$ tracks all the timestamped messages (a_i denotes the time step) received by the automaton during an execution that lead to a state change. First, we show that any two executions with the same run have the same winning identifier.

Observation 1. *If two executions E_1 and E_2 using the same mechanism M have the same finite run, then the winner of both executions has the same player identifier.*

If E_1 and E_2 share the same run and use the same automaton, then they will cause the same state changes because we are using a deterministic automaton, and thus have the same final state s. Based on whether $s\in\mathcal{T}^{\text{fin}}_0$ or $s\in\mathcal{T}^{\text{fin}}_1$, player 0 or 1, respectively, wins in both executions E_1 and E_2.

Next we prove that if the user wins in all executions where the attacker employs a specific strategy (i.e., independent of the environment's strategy), then switching the strategies allows the attacker to win an execution.

Before doing so, we first introduce some notation that allows us to specify a translation from a user strategy to an attacker strategy. Define S^A_{ord} as the attacker's *ordering strategy* that takes the attacker and user messages as input and outputs an ordering of both. Given a user strategy S^U, define the attacker strategy $S^A=(S^U,S^A_{\text{ord}})$ where $\forall e,Y^{\text{U}}: S^A(e,Y^{\text{U}},\sigma)=S^A_{\text{ord}}(S^U(e),Y^{\text{U}})$. Unless specified otherwise, the attacker uses *attacker-first* ordering Ord^A, i.e., $Ord^A(Y^{\text{A}},Y^{\text{U}})=Y^{\text{A}}||Y^{\text{U}}$. Sometimes, we also consider a *user-first* ordering Ord^U, i.e., $Ord^U(Y^{\text{A}},Y^{\text{U}})=Y^{\text{U}}||Y^{\text{A}}$.

Lemma 1. *Let there be a mechanism M, two extended scenarios $X_1=(M,\sigma_1)$, $X_2=(M,\sigma_2)$, and two user strategies S_1, S_2 such that S_1 is playable by U in σ_1 and by A in σ_2 and S_2 is playable by U in σ_2 and by A in σ_1. There exists a reordering strategy $\overline{S^A_{\text{ord}}}$ such that the two attacker strategies $S^A_1=(S_2,Ord^A)$, $S^A_2=(S_1,\overline{S^A_{\text{ord}}})$ satisfy $(Win_{X_1}(S_1,S^A_1)=U)\implies(Win_{X_2}(S_2,S^A_2)=A)$.*

Proof. $Win_{X_1}(S_1,S^A_1)=\text{U}$ means that the user wins in all executions where the attacker's strategy is S^A_1. Consider such an execution $E=(M,\sigma_1,S_1,(S_2,Ord^A),0)$, i.e., the user is assigned 0 and wins. Denote the run of E by r. We should show that there exists at least one execution in the second scenario σ_2 where the attacker wins.

Consider the execution in the second scenario $E'=(M,\sigma_2,S_2,(S_1,\overline{S^A_{\text{ord}}}),1)$, i.e., the attacker is assigned 0 and $\overline{S^A_{\text{ord}}}=Ord^U$ is the user-first ordering strategy. If the mechanism in execution E' does not decide, i.e., E' runs forever, then by definition $Win^{\text{exec}}(E')=\text{A}$ and we are done. Now say the mechanism in E' decides; let its run be r'. We now prove that the two runs are same, i.e., $r=r'$. First, note that in both executions, player 0 uses the strategy S_1 and player 1 uses the strategy S_2. Secondly, since the attacker's reordering strategy prioritizes the attacker

in E and the user in E', the messages sent by 1 are prioritized in both executions. Therefore, the messages the mechanism receives in all time steps are the *same* in both executions including the order of messages. Therefore $r = r'$. Since player 0 (user) wins E, by Observation 1, the attacker wins execution E'. □

Now, in order to bound the number of successful scenarios, we identify scenarios where a mechanism cannot succeed. First, we define *complementary scenarios*, where the players' credential availability is inverted, and show that no mechanism succeeds in both a scenario and its complement.

Definition 8 (Complement scenario). *A scenario $\overline{\sigma}$ is the* complement scenario *of σ if for all i: $\overline{\sigma}_i = \mathsf{stolen}$ if $\sigma_i = \mathsf{safe}$, $\overline{\sigma}_i = \mathsf{safe}$ if $\sigma_i = \mathsf{stolen}$, and $\overline{\sigma}_i = \sigma_i$ otherwise.*

Lemma 2. *If a mechanism succeeds in a scenario σ, then it fails in its complement $\overline{\sigma}$: $Suc(M,\sigma) \implies \neg Suc(M,\overline{\sigma})$.*

The intuition behind the proof is that the attacker will be able to employ in $\overline{\sigma}$ the same messaging strategy that the user uses in σ. Because the user's strategy wins in all executions against any attacker strategy in σ, we show that the attacker's strategy wins in at least one execution of $\overline{\sigma}$. The full proof is in App. D.3.

Next we show that if no credential is safe, then no mechanism succeeds. We call such scenarios *bad*.

Lemma 3. *Given a bad scenario $\sigma^{\text{bad}} \in \{\mathsf{lost}, \mathsf{leaked}, \mathsf{stolen}\}^n$, where all credentials are unsafe, $\nexists M \in \mathcal{M}_n$: $Suc(M,\sigma^{\text{bad}})$.*

We prove this by contradiction, i.e., say the user wins in σ^{bad}. Using the winning strategy of the user, we show the existence of a winning strategy for the attacker, thus arriving at a contradiction. We defer the proof to App. D.3.

4.2 Profile size bound

We can now bound the size of feasible security profiles using Lemma 2 and Lemma 3.

Theorem 1. *The maximum number of scenarios an n-credential mechanism succeeds is*

$$P(n) = (4^n - 2^n)/2, \tag{1}$$

i.e., $\forall M \in \mathcal{M}_n : |prof(M)| \le P(n)$.

For proving, we first identify four subsets of scenarios, which will also be useful later. Define a *with-safe scenario* (resp., *with-stolen scenario*) as a scenario in which at least one credential's state is safe (resp., stolen). Denote the set of all with-safe (resp., with-stolen) scenarios by Σ_{ws} (resp., Σ_{wt}; we use the second letter, 't'). Define a *no-safe scenario* (resp., *no-stolen scenario*) as a scenario in which no credential's state is safe (resp., stolen). Denote the set of all no-safe (resp., no-stolen) scenarios by Σ_{ns} (resp., Σ_{nt}). The number of no-safe scenarios is $|\Sigma_{\text{ns}}| = 3^n$ and all mechanisms fail in all of them (Lemma 3). Denote by $\overline{T}$ the complement set of T with respect to all scenarios. Note that $\overline{\Sigma_{\text{ws}}} = \Sigma_{\text{ns}}$ and $\overline{\Sigma_{\text{wt}}} = \Sigma_{\text{nt}}$.

Denote with-safe-with-stolen scenarios by $\Sigma_{\text{wswst}} = \Sigma_{\text{ws}} \cap \Sigma_{\text{wt}}$; these play a special role in our proofs, so we count them now. By De-Morgan's law, $\overline{\Sigma_{\text{wswst}}} = \Sigma_{\text{ns}} \cup \Sigma_{\text{nt}} = |\Sigma_{\text{ns}}| + |\Sigma_{\text{nt}}| - |\Sigma_{\text{ns}} \cap \Sigma_{\text{nt}}|$. Substitute $|\Sigma_{\text{ns}}| = |\Sigma_{\text{nt}}| = 3^n$ and $|\Sigma_{\text{ns}} \cap \Sigma_{\text{nt}}| = 2^n$ (as each credential is either leaked or lost). So we have

$$|\Sigma_{\text{wswst}}| = |\Sigma| - |\overline{\Sigma_{\text{wswst}}}| = 4^n - 2 \cdot 3^n + 2^n. \tag{2}$$

We can now prove Theorem 1.

Proof. If $n = 1$, there are 4 scenarios in total and all mechanisms fail in the 3 unsafe ones (Lemma 3). Hence $P(1) = 1$.

For $n \ge 2$, observe that for each with-safe-with-stolen scenario $\sigma \in \Sigma_{\text{wswst}}$ (which exist for all $n \ge 2$) that a mechanism $M \in \mathcal{M}_n$ succeeds in, there is a complement $\overline{\sigma}$ where it fails (Lemma 2). Crucially, $\overline{\sigma} \in \Sigma_{\text{wswst}}$ because a with-safe-with-stolen scenario has (at least) one safe and one stolen credential, and the safe, stolen states are switched in the complement. Therefore, M cannot succeed in more than half with-safe-with-stolen scenarios. The number of with-safe scenarios where all mechanisms fail is lower-bounded by $Q_{\text{ws}}(n) = |\Sigma_{\text{wswst}}|/2 \overset{\text{eq.(2)}}{=} \frac{4^n - 2 \cdot 3^n + 2^n}{2}$.

The number of scenarios in which all mechanisms fail is lower-bounded by $Q(n) = Q_{\text{ws}}(n) + |\Sigma_{\text{ns}}| = \frac{4^n - 2 \cdot 3^n + 2^n}{2} + 3^n = \frac{4^n + 2^n}{2}$. Hence, the number of scenarios in which any mechanism succeeds is bounded by $P(n) = |\Sigma| - Q(n) = 4^n - \frac{4^n + 2^n}{2} = \frac{4^n - 2^n}{2}$. □

5 MAXIMAL MECHANISMS

We now specify an approach to generate maximal mechanisms. These mechanisms wait for a bounded time allowing both players to submit credentials, therefore called *bounded-delay mechanisms*.

Any player can initiate authentication with a bounded-delay mechanism, which then starts a timer. Both U and A can send messages carrying credentials until the time elapses. Say the set of credentials submitted by player 0 is C_0 and player 1 is C_1. A deterministic *judging function* J selects the winner: It takes the two sets of credentials sent by the players and outputs an identifier, i.e., $J(C_0, C_1) \mapsto \{0, 1\}$. Bounded-delay mechanisms are realizable through the automaton model in a straightforward manner. Details are in App. C. Given a judging function J, we denote the corresponding mechanism by $M(J)$.

We present three properties of judging functions (§5.1) and show that they are sufficient to produce a maximally secure mechanism (§5.2). Then, we present three functions that produce, for any n, maximal mechanisms, along with some non-bounded-delay maximal mechanisms (§5.3).

5.1 Well-formed judging functions

Any bounded-delay mechanism with judging functions satisfying the following properties are maximal.

Definition 9 (ID-Agnostic (IA)). *A judging function J is* ID-Agnostic *if the identifier assignment does not affect the result, i.e., if $C \ne C'$ then $(J(C,C') = 0) \Leftrightarrow (J(C',C) = 1)$.*

If a judging function J satisfies IA, then we can compare any two sets of credentials and say that one of them is better than the other. We use the notation $C >_J C'$ (subscript omitted when obvious) to mean that C is *better* than C' according to J. Similarly, $C \ge_J C'$ means that either the two credential sets are the same ($C = C'$) or C is better than C'.

Note that we do not say anything about the case where $C = C'$ in the above definition, so an ID-Agnostic function can choose

to output either 0 or 1. WLOG all functions we present follow $\forall C: J(C,C) = 0$.

Definition 10 (Knowledge-rewarding (KR)). *A judging function is* knowledge rewarding *if, when the credentials submitted by one player are a strict subset of those submitted by the other, then it returns the latter, i.e., if $C' \subset C$ then $C' \prec C$.*

Definition 11 (Transitive knowledge-rewarding (TKR)). *A judging function is* transitive knowledge rewarding *if additional credentials cannot weaken a player's strategy, i.e., if $C_0 \neq C_1$, $C_1 \neq C_2$, $C_0 \succ C_1$, and $C_1 \succ C_2$, then $C_0 \not\subseteq C_2$.*

Note that $C_0 \neq C_2$ is implied above because if instead $C_0 = C_2$, then C_1 is both better and worse than C_0, which is not possible for an ID-Agnostic judging function.

A judging function with all the three properties and its resultant mechanism are *well formed.*

Definition 12 (Well-formedness). *If a judging function J satisfies IA, KR, and TKR then it is* well-formed*, and the resulting bounded-delay mechanism $M(J)$ is* well-formed.

Note that a well-formed judging function J need not be *transitive*, i.e., it could allow: $C_0 \succ_J C_1$, $C_1 \succ_J C_2$ and $C_2 \succ_J C_0$. Examples of such functions appear later.

5.2 Maximality of well-formed mechanisms

We now prove that any well-formed mechanism is maximal. Two lemmas do a bulk of the work: Lemma 5 shows that the user can win in all with-safe-no-stolen scenarios and Lemma 8 shows that the user can win in exactly half of the with-safe-with-stolen scenarios. Using these results, Lemma 9 shows that the profile size of any n-credential well-formed mechanism is $P(n)$, which in turn implies maximality.

Before proving when well-formed algorithms succeed, we define submit-early strategies and prove Lemma 4, which we use throughout the rest of the proofs.

Definition 13 (Submit-early strategy). *A* submit-early strategy *involves sending a single credential set when the automaton is in its initial state and nothing thereafter.*

The next lemma says that if a player employs a submit-early strategy (Definition 13) with credentials C, the opponent can only win by submitting a better or equal credential set.

Lemma 4. *Given a well-formed bounded-delay mechanism M and a scenario σ, if one player employs a submit-early strategy with credential set C, but the other player wins an execution, then the winning player must have submitted better or equal credentials C', i.e., $C' \succeq C$.*

Proof. Say player 0 submits credentials C as part of a submit-early strategy, but player 1 wins an execution. Denote the final set of credentials submitted by player 1 (perhaps across multiple messages) as C'.

Two cases emerge based on the order in which the two sets of credentials, C and C', are processed by the automaton. But, irrespective of the order, player 1 wins only if $C' \succeq C$: IA guarantees that, if on the contrary $C' \prec C$, then the automaton would not prefer player 1 irrespective of the player identifier (0 or 1) assigned to it. So player 1 cannot win an execution, and therefore, it must be that $C' \succeq C$. □

Denote by S^U_{all} (resp., S^A_{all}) the submit-early strategy where all credentials owned by the user C^U_σ (resp., attacker C^A_σ) are submitted. (S^A_{all} uses attacker-first ordering, as is the case whenever we do not explicitly specify it.)

We now prove the first major lemma.

Lemma 5. *Well-formed mechanisms succeed in with-safe-no-stolen scenarios, i.e., for all well-formed mechanisms M and scenarios $\sigma \in \Sigma_{\text{wsnt}}$: $Suc(M,\sigma) = \mathit{True}$.*

Proof. Assume for contradiction that the mechanism fails in a with-safe-no-stolen scenario $\sigma \in \Sigma_{\text{wsnt}}$. This means that the attacker wins at least one execution when the user follows the submit-early strategy S^U_{all} where it submits all its credentials C^U_σ. Then, due to Lemma 4, the attacker must know a set of credentials $C \subseteq C^A_\sigma$ such that $C \succeq C^U_\sigma$. And since the scenario σ is a with-safe-no-stolen scenario, we have $C^A_\sigma \subset C^U_\sigma$, and since $C \subseteq C^A_\sigma$, we conclude that $C \subset C^U_\sigma$.

But due to the knowledge-rewarding (KR) property, $C \subset C^U_\sigma$ implies $C \prec C^U_\sigma$, thereby contradicting $C \succeq C^U_\sigma$. Thus the attacker cannot win an execution in a with-safe-no-stolen scenario and the mechanism succeeds. □

Our next goal is to prove that the user wins in half of the with-safe-with-stolen scenarios Σ_{wswst}. Our approach is as follows. Recall that for every with-safe-with-stolen scenario σ, it's complement $\overline{\sigma}$ is also a with-safe-with-stolen scenario. Since any mechanism fails in one of σ or $\overline{\sigma}$ (Lemma 2), we use a winning attacker strategy in σ or $\overline{\sigma}$ to derive a winning user strategy in the other scenario.

First, we show that if an attacker winning strategy exists for an execution, then the submit-early strategy with all credentials also wins for that execution. Let S^A_{all} define the submit-early strategy for the attacker where it submits all its credentials and uses attacker-first ordering.

Lemma 6. *Given a well-formed mechanism M, let the winning strategy of an attacker in a with-safe-with-stolen scenario $\sigma \in \Sigma_{\text{wswst}}$ be $S^A \neq S^A_{\text{all}}$, then S^A_{all} is also a winning strategy. That is, if $X = (M,\sigma)$ and $\forall S^U : Win_X(S^U, S^A) = A$ then $\forall S^U : Win_X(S^U, S^A_{\text{all}}) = A$.*

The proof, which follows from the KR and the TKR properties, is deferred to App. D.3. Now, we prove that $\bar{S}^U_{\text{all}}$ is a winning strategy for the user in the complement scenario $\overline{\sigma}$.

Lemma 7. *If S^A_{all} is a winning strategy for the attacker in a with-safe-with-stolen scenario $\sigma \in \Sigma_{\text{wswst}}$, then the user strategy $\bar{S}^U_{\text{all}}$ is a winning strategy for the user in the complement with-safe-with-stolen scenario $\overline{\sigma}$.*

We defer this proof to App. D.3; again, it follows from the IA and TKR properties. Now we can prove the second major lemma.

Lemma 8. *For all well-formed mechanisms M and with-safe-with-stolen scenarios $\sigma \in \Sigma_{\text{wswst}}$, M succeeds in either σ or in its complement $\overline{\sigma}$, i.e., $Suc(M,\sigma) \vee Suc(M,\overline{\sigma})$.*

Proof. For each scenario $\sigma \in \Sigma_{\text{wswst}}$, its complement $\overline{\sigma}$ satisfies $\overline{\sigma} \neq \sigma$ and $\overline{\sigma} \in \Sigma_{\text{wswst}}$. Consider a pair of complement scenarios $\sigma, \overline{\sigma} \in \Sigma_{\text{wswst}}$. No well-formed mechanism M succeeds in both σ and $\overline{\sigma}$ (Lemma 2). WLOG assume that M fails in σ, i.e., there exists a winning strategy S^A for the attacker that allows it

Algorithm 2 Priority judging functions: Priority (PR) J^V_{pr} and Priority with exception (PRE) J^V_{pre}. Highlighted lines only for PRE.

Require: V is a permutation over the elements of the set C_{all}.
function $J^V(C_0,C_1)$ ▷ $C_0 \subseteq C_{\text{all}}, C_1 \subseteq C_{\text{all}}$
  if $C_0 = \{V_{n-1}\} \wedge C_1 = \{V_n\}$ **then return** 1 ▷ Only for J^V_{pre}
  if $C_0 = \{V_n\} \wedge C_1 = \{V_{n-1}\}$ **then return** 0 ▷ Only for J^V_{pre}
  for $c = V_1, V_2, \ldots, V_n$ **do**
    if $c \in C_0 \wedge c \notin C_1$ **then return** 0 ▷ $C_0 > C_1$
    if $c \in C_1 \wedge c \notin C_0$ **then return** 1 ▷ $C_1 > C_0$
  return 0 ▷ Default

to win an execution in σ. By Lemma 6, the existence of a winning strategy S^A implies that the submit-early strategy S^A_{all} is also a winning strategy. And by Lemma 7, $\bar{S}^U_{\text{all}}$ is a winning strategy for the user in the complement scenario $\overline{\sigma}$. □

We conclude by proving that the profile size of a well-formed mechanism is $P(n)$.

Lemma 9. *The profile size of any well-formed n-credential mechanism is $P(n)$.*

Proof. Lemma 5 shows that a well-formed mechanism succeeds in all with-safe-no-stolen scenarios ($\Sigma_{\text{wsnt}} = \Sigma_{\text{ws}} \cap \Sigma_{\text{nt}}$). By basic set theory, $|\Sigma_{\text{wsnt}}| = |\Sigma_{\text{nt}}| - |\Sigma_{\text{nt}} \setminus \Sigma_{\text{ws}}|$ and $|\Sigma_{\text{nt}} \setminus \Sigma_{\text{ws}}| = |\Sigma_{\text{nt}} \cap \overline{\Sigma_{\text{ws}}}| = |\Sigma_{\text{nt}} \cap \Sigma_{\text{ns}}|$. Finally, since $|\Sigma_{\text{nt}}| = 3^n$ and $|\Sigma_{\text{nt}} \cap \Sigma_{\text{ns}}| = 2^n$, we have $|\Sigma_{\text{wsnt}}| = 3^n - 2^n$.

Lemma 8 shows that a well-formed mechanism succeeds in exactly half of the with-safe-with-stolen scenarios ($\Sigma_{\text{wswst}} = \Sigma_{\text{ws}} \cap \Sigma_{\text{wt}}$). From eq. (2), $|\Sigma_{\text{wswst}}| = (4^n - 2 \cdot 3^n + 2^n)$.

In total, a well-formed mechanism succeeds in $|\Sigma_{\text{wsnt}}| + |\Sigma_{\text{wswst}}|/2 = (4^n - 2^n)/2 = P(n)$ scenarios. □

Corollary 1. *A well-formed mechanism is maximally secure.*

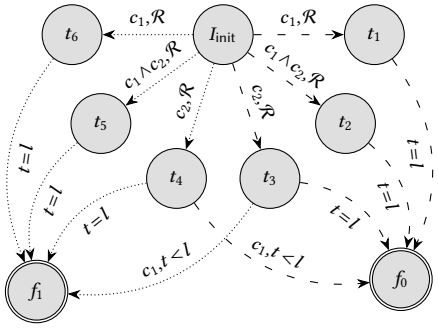
This is straightforward because the existence of a mechanism better than a well-formed mechanism (Lemma 9) violates the profile size bound (Theorem 1).

5.3 Algorithms for maximal mechanisms

We now specify three algorithms that produce maximal n-credential mechanisms for any n.

5.3.1 Priority mechanisms. Let a vector V define an ordering over all the credentials, then the priority judging function using V (Algorithm 2, uncolored) decides on the player that submits a unique high-priority credential.

Denote the priority mechanism corresponding to the judging function J^V_{pr} by M^V_{pr}. For example, the 2-credential priority mechanism $M^{[c_1,c_2]}_{\text{pr}}$ along with its profile is in Fig. 1. Note that player 0 (resp., 1) can only take dashed (resp., dotted) edges and tries to reach f_0 (resp., f_1), and $\mathcal{R}$ denotes a clock reset. A proof for the profile computation is in App. D. Note that for any n, Algorithm 2 yields exactly one distinct mechanism because changes in the permutation V yield equivalent mechanisms, e.g., $M^{[c_1,c_2]}_{\text{pr}} \cong M^{[c_2,c_1]}_{\text{pr}}$.



c_1 \ c_2	Stolen	Leaked	Lost	Safe
Stolen	0	0	0	0
Leaked	0	0	0	1
Lost	0	0	0	1
Safe	1	1	1	1

Figure 1: $M^{[c_1,c_2]}_{\text{pr}}$ and its profile.

Multi-timeouts. It is also possible to implement a priority mechanism using a timeout-based approach. Such mechanisms leverage multiple timeouts and are hence not bounded-delay (This does not impact our formalization as multiple clocks can be removed [7]).

We illustrate the idea with a 2-credential mechanism. The mechanism defines two time thresholds, x_1 and x_2 for credentials c_1 and c_2 respectively. It uses timers t_1 and t_2 to track the time elapsed since credential c_1 and c_2 were submitted. The mechanism ends when $t_1 = x_1$ or $t_2 = x_2$. Setting $x_1 < x_2$ effectively prioritizes c_1 over c_2.

Priority mechanisms can also be constructed by mixing the two aforementioned approaches. For example, a 4-credential priority mechanism can be designed by combining the 2-credential mechanism given in Fig. 1 with a timeout of x_1 and a 2-credential multi-timeout mechanism with timeouts x_2 and x_3 where $x_1 < x_2 < x_3$.

Multi-timeout-mechanisms are suited in settings with asymmetric credentials where credentials require different timeouts. Details about multi-timer priority mechanisms are in App. B.

5.3.2 Priority with exception. Mechanisms in this category are similar to priority mechanisms except when the last two credentials in the priority rule are submitted by the two players. For example, if the priority vector is $V = [c_2, c_3, c_1]$, then the exception is $\{c_1\} > \{c_3\}$. The resultant judging function is denoted by J^V_{pre} and is specified in Algorithm 2. Like with priority mechanisms, this algorithm yields at most one maximal mechanism.

5.3.3 Majority mechanisms. Mechanisms in this category favor the player submitting the *most credentials*, so $C > C'$ if $|C| > |C'|$. To break ties, different strategies are possible which we model through a tie-breaking function $T(C_0, C_1) \mapsto \{0, 1\}$. We only consider ID-Agnostic tie-breaking functions, i.e., if $0 \leftarrow T(C_0,C_1)$ then $1 \leftarrow T(C_1,C_0)$. The resultant judging function is denoted J^T_{maj} and is specified in Algorithm 3. The profile of a majority mechanism using the priority vector $[c_1,c_2,c_3]$ to break ties is in Tab. 1. Note that it differs from the profile of a priority mechanism using the same priority vector (Tab. 3) only in two scenarios, namely, the scenario (c_1 safe, c_2 and c_3 stolen) and its complement.

c_1 **safe**

c_2 \ c_3	St	Le	Lo	Sa
St	0	1	1	1
Le	1	1	1	1
Lo	1	1	1	1
Sa	1	1	1	1

c_1 **lost**

c_2 \ c_3	St	Le	Lo	Sa
St	0	0	0	0
Le	0	0	0	1
Lo	0	0	0	1
Sa	1	1	1	1

c_1 **leaked**

c_2 \ c_3	St	Le	Lo	Sa
St	0	0	0	0
Le	0	0	0	1
Lo	0	0	0	1
Sa	1	1	1	1

c_1 **stolen**

c_2 \ c_3	St	Le	Lo	Sa
St	0	0	0	0
Le	0	0	0	0
Lo	0	0	0	0
Sa	0	0	0	1

Table 1: Profile of a majority 3-cred mechanism $M_{\text{maj}}^{[c_1,c_2,c_3]}$

Unlike the two priority-based mechanisms, this algorithm yields many distinct maximal mechanisms for all $n > 2$. The total number of majority mechanisms including credential permutations is $q(n) = 2^{\binom{2n-1}{n-1}-2^{n-1}}$ (App. D). For $n = 3$, we manually discard equivalent mechanisms to find a total of 12 distinct majority mechanisms (out of $q(3) = 64$).

Algorithm 3 The majority judging function J_{maj}^{T}

function $J_{\text{maj}}^{T}(C_0, C_1)$ ▷ $C_0 \subseteq C_{\text{all}}, C_1 \subseteq C_{\text{all}}$
  if $|C_0| > |C_1|$ **then return** 0
  if $|C_1| > |C_0|$ **then return** 1
  return $T(C_0, C_1)$ ▷ If $|C_0| = |C_1|$

6 COMPLETE MAXIMAL SETS

A *complete maximal mechanism set* is a minimal set of n-credential mechanisms such that *any other n-credential mechanism is either equivalent to or worse than some mechanism in this set.*

Definition 14 (Complete set). *A complete maximal mechanism set of n-credential mechanisms $\mathcal{O} = \{M_1, M_2 \ldots\} \subset \mathcal{M}_n$ satisfies three properties: (1) each $M \in \mathcal{O}$ is maximal; (2) any two distinct members are incomparable, i.e., $\forall M, M' \in \mathcal{O}, M \neq M' : M \nsim M'$; and (3) any n-credential mechanism is worse or equivalent to some mechanism in $\mathcal{O}$, i.e., $\forall M \in \mathcal{M}_n : \exists M' \in \mathcal{O}$ s.t. $M \preceq M'$.*

Complete maximal sets are important because one of the mechanisms in this set has the highest probability of succeeding in any real-world setting, i.e., for any given probability distribution of credential states. Several different complete maximal sets exist, but all their sizes are the same as each mechanism in a maximal set will have an equivalent in the other (See App. D).

We now present complete maximal sets for all $n \leq 3$. These sets are composed of bounded-delay mechanisms. For one-credential mechanisms, the attacker wins in three of the four possible scenarios (Lemma 3). Hence, the priority mechanism $M_{\text{pr}}^{[c_1]}$ is the only one in the complete maximal set $\mathcal{O}_1$.

The complete maximal set of 2-credential mechanisms is of size one. We find a set with a priority mechanism.

Theorem 2. *A complete maximal set of 2-credential mechanisms is $\mathcal{O}_2 = \{M_{\text{pr}}^{[c_1,c_2]}\}$.*

Proof. We prove $\mathcal{O}_2$ satisfies the three requirements (Definition 14). The first is that $M_{\text{pr}}^{[c_1,c_2]}$ is maximal, which follows from Lemma 10. The second is that every pair of mechanisms in $\mathcal{O}_2$ must be incomparable to each other, which is vacuously true.

The third is that any 2-credential mechanism M must satisfy $M \preceq M_{\text{pr}}^{[c_1,c_2]}$. Recall that $M \preceq M_{\text{pr}}^{[c_1,c_2]}$ implies the existence of a mechanism $M' \cong M_{\text{pr}}^{[c_1,c_2]}$ such that $prof(M) \subseteq prof(M')$. There are only two choices for M' that yield distinct profiles, namely $M' \in \{M_{\text{pr}}^{[c_1,c_2]}, M_{\text{pr}}^{[c_2,c_1]}\}$. And we are trying to prove that for any M, one of $prof(M) \subseteq prof(M_{\text{pr}}^{[c_1,c_2]})$ or $prof(M) \subseteq prof(M_{\text{pr}}^{[c_2,c_1]})$ is true.

We prove by contradiction. Assume there exists a mechanism M such that $prof(M) \nsubseteq prof(M_{\text{pr}}^{[c_1,c_2]})$ and $prof(M) \nsubseteq prof(M_{\text{pr}}^{[c_2,c_1]})$.

Consider the set of seven with-safe scenarios $\Sigma_{\text{ws}} = \{\hat{\sigma}_1, \ldots, \hat{\sigma}_7\}$. Without loss of generality, let $\hat{\sigma}_1$ = (safe, stolen) (i.e., c_1 safe, c_2 stolen) and $\hat{\sigma}_7$ = (stolen, safe) is the complement of $\hat{\sigma}_1$. Figure 1 shows that $prof(M_{\text{pr}}^{[c_1,c_2]}) = \{\hat{\sigma}_1, \ldots, \hat{\sigma}_6\}$ and $prof(M_{\text{pr}}^{[c_2,c_1]}) = \{\hat{\sigma}_2, \ldots, \hat{\sigma}_7\}$.

By Lemma 3, the profile of M is a subset of Σ_{ws}, i.e., $prof(M) \subseteq \Sigma_{\text{ws}}$. By basic set theory, the only way to guarantee $prof(M) \nsubseteq prof(M_{\text{pr}}^{[c_1,c_2]})$ and $prof(M) \nsubseteq prof(M_{\text{pr}}^{[c_2,c_1]})$ is if $\{\hat{\sigma}_1, \hat{\sigma}_7\} \in prof(M)$. But Lemma 2 rules this out as they are complement scenarios. □

The complete maximal set of 3-credential mechanisms is of size 14. We group the constituents into majority and priority mechanisms, denoted by $\mathcal{O}_{\text{maj},3}$ and $\mathcal{O}_{\text{pr},3}$ respectively, i.e., $\mathcal{O}_3 = \mathcal{O}_{\text{maj},3} \cup \mathcal{O}_{\text{pr},3}$. The set $\mathcal{O}_{\text{pr},3}$ contains two priority-based mechanisms: the regular one (Algorithm 2) and with an exception (Algorithm 2).

The set $\mathcal{O}_{\text{maj},3}$ contains 12 majority mechanisms that differ in their tie-breaking rule. Two tie rules are possible: linear priority, e.g., $c_1 \succ c_2, c_2 \succ c_3, c_1 \succ c_3$, or cyclic priority, e.g., $c_1 \succ c_2, c_2 \succ c_3, c_3 \succ c_1$ (last one switched). Note that a cyclic rule makes the resultant judging function non-transitive. The twelve majority mechanisms differ in the choice of tie rule used to break ties between 1-credential and 2-credential sets, e.g., one of them uses a linear rule for 1-credential sets and a cyclic rule for 2-credential sets. More details are in App. E.

7 APPLICATIONS

We now apply our framework to analyze a popular cryptocurrency wallet (§7.1), a bank account (§7.2) and all known 2-credential mechanisms (§7.3).

7.1 The Argent Cryptocurrency Wallet

Social recovery is a prominent approach [22] to design non-custodial cryptocurrency wallets where a user's social circle, e.g., friends and family, is used to manage keys. We analyze Argent [6], a popular social-recovery wallet [12]. We describe Argent's operation as an automaton (§7.1.1), find the resultant profile (§7.1.2) and present security improvements (§7.1.3).

7.1.1 Operation. The Argent mechanism M_{Arg}^{m} uses $m + 1$ credentials, consisting of an owner credential o and a set of m so-called *guardian* credentials $\mathcal{G} = \{g_1, g_2, \ldots, g_m\}$. The owner credential is a cryptographic key on the user's mobile phone.

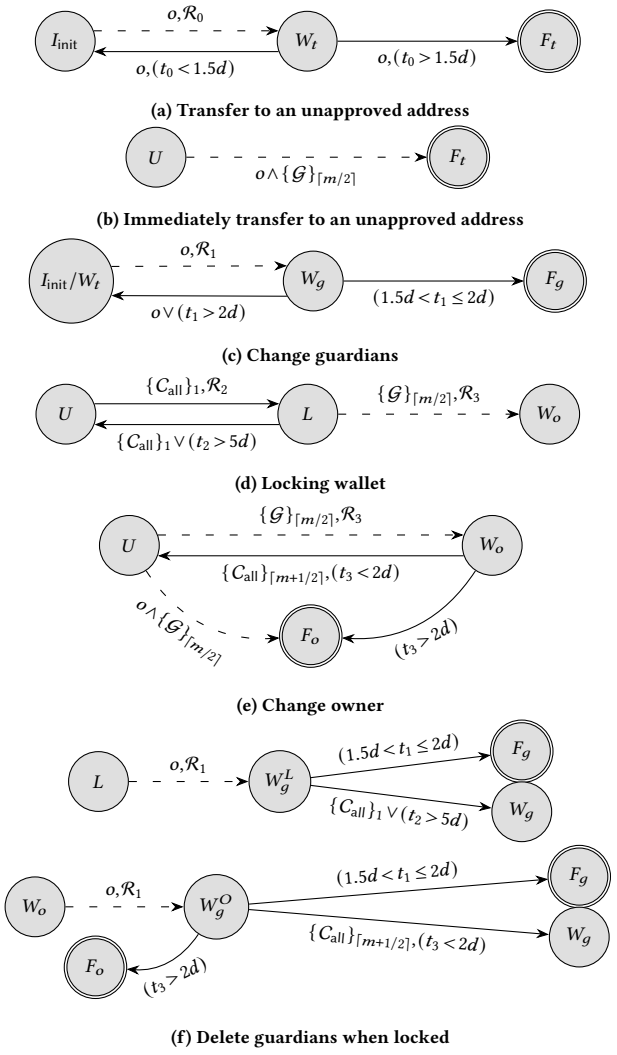


Figure 2: Argent mechanism M^m_{Arg} sub-automata

Guardians can be anything from a friend's cryptocurrency wallet to a hardware wallet or a paid third-party service. Thus, in our notation, $C_{\text{all}} = \{o, g_1, g_2, \ldots, g_m\}$.

The user maintains a list of *approved addresses* (called *trusted contacts*) to which she can send funds immediately. Transferring funds to a unapproved address requires additional steps. The Argent wallet allows adding or removing approved addresses, adding or removing guardians, replacing the owner credential (if her phone is lost or stolen) and transferring deposited funds.

Figure 2 depicts several automata, each capturing a different functionality of the Argent mechanism. The state I_{init} is the starting state of the automaton. Dashed edges indicate an edge that can only be taken by player 0 whereas solid edges can be taken by either player. We only show half of the automaton, the portion containing the winning states for 0; we omit the other half because it is symmetric.

We consider any action that withdraws money or changes the set of credentials to be sensitive and mark them as final. Three final states exist: F_t (transfer money), F_g (add / remove guardians) and F_o (replace owner). We model credential changes as final states because in the case of Argent, the ability to change credentials (against all opponent's strategies) is powerful enough (details in App. F.1.1).

Note that we use multiple clocks to simplify the presentation: in particular, $\mathcal{R}_i$ denotes a reset of ith clock, i.e., $t_i \leftarrow 0$.

There are two ways for a player to withdraw funds to an address it controls without changing the credentials (Fig. 2a, Fig. 2b). She can add this address to the approved list and then withdraw, a process we call *slow withdrawal*. In order to do so, the owner needs to initiate an action (I_{init} to W_t). The addition becomes effective after 1.5 days, at which point the owner can withdraw (W_t to F_t). The approval process can be cancelled by the owner during the 1.5 day period (W_t to I_{init}): this is useful if the owner credential o is leaked.

Another way is to order a *fast withdrawal* with the owner and at least half of the guardians (Fig. 2b).

Argent allows changing the set of guardians (Fig. 2c). The owner initiates the process (I_{init} to W_g) and can be finalized only during a time window that starts 1.5 days after initiation and lasts for 12 hours thereafter (W_g to F_g). As in the slow withdrawal case, the guardian change process can be cancelled by o.

Argent implements *locking*, a feature intended for situations when the owner suspects a credential fault [6]. As shown in Fig. 2d, the mechanism can be moved from any state into a locked state with any one credential. A limited set of actions are possible in the locked state, namely, change owner and unlock. Unlocking the automaton, i.e., moving back into its previous state can be done with any credential. The state U represents all unlocked states, i.e., the initial state I_{init}, or the waiting states W_t or W_g.

Argent allows changing the owner, a process they call *recovery*, in two ways (Fig. 2e). If the current owner credential is lost, the automaton can be moved from any unlocked state to the recovery state W_o with $\lceil m/2 \rceil$ guardian credentials. The new owner address is specified in this step. Finalizing this new owner takes 2 days (W_o to F_o). In this period, owner change can be cancelled with $\lceil (m+1)/2 \rceil$ credentials, which can include the original owner: this is useful if the owner credential was not actually lost. Note that Argent doesn't treat the state W_o as an unlocked state, i.e., it is not part of U.

The second way is for when the owner credential is safe, e.g., if the user wants to transfer the Argent app between phones. In this case, there is no need to wait. The owner can be changed immediately with the owner credential o and $\lceil m/2 \rceil$ guardians, a process we call *fast owner-change*.

Finally, Argent allows guardian revocation (but not addition) from a locked (or recovery) state (Fig. 2f). Confirming revocation can be done after 1.5 days like before, but canceling it requires unlocking (or canceling recovery) first.

7.1.2 Profile analysis. We now analyze the security of Argent M^m_{Arg} with one and with two guardians ($m = 1,2$).

The one guardian case has just two credentials: an owner o and a guardian g_1. If one of the credentials is lost but the other is safe, then the user wins by revoking the lost credential.

If o is safe and g_1 is stolen, the user wins by revoking the unsafe guardian credential (from any state). In particular, even if the attacker initiates an owner change before (I_{init} to W_o), the user can revoke guardian (reach W_g^O in Fig. 2f). A similar strategy lets the user win if o is safe and g_1 is leaked.

If g_1 is safe and o is leaked or stolen, the mechanism fails and the attacker's winning strategy is to lock the automaton if the current state is unlocked, and cancel a recovery if the current state is W_o (i.e., owner change was initiated).

In summary, the profile of 1-guardian Argent has 5 scenarios, and is hence weaker than our maximal mechanism with 6 scenarios, i.e., $M^1_{\text{Arg}} \prec M^{[o,g_1]}_{\text{pr}}$ (Fig. 1). We similarly analyze 2-guardian Argent in App. F to find that its profile has 22 scenarios, and that it is worse than a priority mechanism with 28 scenarios, i.e., $M^2_{\text{Arg}} \prec M^{[o,g_1,g_2]}_{\text{pr}}$.

7.1.3 Improving Argent. We propose a simple strategy to improve Argent's profile: executing multiple transactions atomically, commonly known as a multicall [2]. For example, consider the scenario (o leaked, g_1 safe) where Argent previously failed. It succeeds now because the user can atomically execute two transitions: unlock and fast withdrawal from a locked state (L or W_o). With a multicall, Argent becomes maximal with 1 guardian, but not with 2 guardians or more. For example, with two guardians, the profile of Argent with multicalls has 24 scenarios, weaker than 28 in a maximal mechanism.

While it is technically feasible to run a multicall with Argent's contracts today (e.g., using Uniswap's contract [2]), we could not find a mention of this technique in Argent's documentation and Argent does not natively support it.

As noted before, the use of multicalls only helps improve Argent's profile to an extent. One can attain better security with our maximal mechanisms to achieve the same functionality as Argent, except we omit locking as it does not improve security in our model. Rather than using different mechanisms for the various functions, e.g., change owner / guardian, we propose the use of a maximal mechanism for all functionalities, thus vastly simplifying the design. We propose the use of a majority mechanism with some tie-breaking function due to its simplicity.

One missing feature from our proposed design is fast withdrawals (Fig. 2b) or fast owner change (Fig. 2e). However, it is easy to add it: simply move to the final state if *all the credentials* are submitted. The modified mechanism is still maximal, i.e., doesn't break PA, KR and TKR, since the fast path can only be enabled if all the credentials are provided. However, it does incur a usability hit: for all $m > 1$, the user needs to do more work in gathering guardian approvals than with Argent. (If $m = 1$, the user needs to submit all the credentials anyway.)

We demonstrate the practicality of our mechanisms with a proof-of-concept priority mechanism in Solidity (App. A). Our implementation takes about 100 lines of code and requires 210k gas for the costliest function, compared to 150k for Argent ($7.6 and $5.4 resp., assuming a gas price of 30gwei and Ethereum price of $1200).

7.2 HDFC Online Bank Account

We model the authentication mechanisms used by the HDFC bank website [3]. The bank provides a web portal for the users to login with their password credential c_p. We assume that 2FA is enabled, in particular, one time PINs are received on the registered mobile number; this credential is denoted c_m. The bank allows users to add additional factors if needed, e.g., email, but we do not model it for simplicity. Finally, users can change the registered mobile number by submitting a request at the bank[1] along with an ID proof [5]. We could not find information on whether the web account is locked during this process in the bank's documentation. We assume that no such locked state exists for our analysis. We model any acceptable identity proof using the credential c_{id}.

HDFC

c_m \ c_p	leaked	lost	safe
stolen	0	0	0
lost	1	1	1
safe	1	1	1

HDFC improved

c_m \ c_p	leaked	lost	safe
stolen	0	0	1
lost	1	1	1
safe	1	1	1

Table 2: Profile of the HDFC mechanism and our proposed improvement in probable scenarios. c_{id} is safe.

Like before, we discuss how to withdraw money to a new account number and how to change credentials. Transferring money involves using the password c_p to login and authenticating via mobile c_m to initiate the third-party addition process. This process takes 30 minutes, during which it can be cancelled by logging in to the bank portal.[2] After the time elapses, money can be transferred to the new account. This requires another 2FA.

The bank allows changing the password after authenticating via mobile. In order to change the registered mobile number, the user must submit a form along with relevant identity proof c_{id}, and the change happens after 3 days [4].

7.2.1 Profile analysis. Before analyzing the profile, we discuss probable scenarios based on the nature of credential vulnerabilities. We assume that ID proofs are predominantly safe. The primary risks for passwords (c_p) involve either being lost or leaked (for instance, through weak password practices or breaches), whereas mobile credentials (c_m) are chiefly at risk of being lost or stolen (such as via SIM swaps).

We now evaluate HDFC's profile in these scenarios. The mechanism succeeds irrespective of the state of c_p when c_m is safe or lost. In the former case (c_m safe), the user's winning strategy is to change the password. Whereas in the latter (c_m lost), the winning strategy is to change the mobile number by physically submitting the ID credential. However, if c_m is stolen (or leaked), the mechanism fails because the three day delay is enough for the attacker to withdraw money. Like in the case of Argent, a win for the user implies that the user will be able to transfer money. In total, HDFC's profile has 20 scenarios. A subset of its complete profile in the likely scenarios is in Tab. 2. The automaton and a complete profile is in App. F.

7.2.2 Improving HDFC. We propose a minor yet effective modification to HDFC's existing authentication mechanism: use a two-credential priority mechanism between c_m and c_p, prioritizing c_p over c_m, for withdrawals and password resets. This adjustment, while keeping the rest of the mechanism involving the ID credential unchanged, yields a significant improvement in security. It effectively transforms HDFC into a multi-timeout

[1] There are other ways to change the mobile number [4], e.g., use a debit card and visit an ATM. We do not model these for simplicity.

[2] The delay is only present when transferring funds to a new account. Transfers to previously added accounts are instantaneous. We assume that an attacker's account number was not previously added.

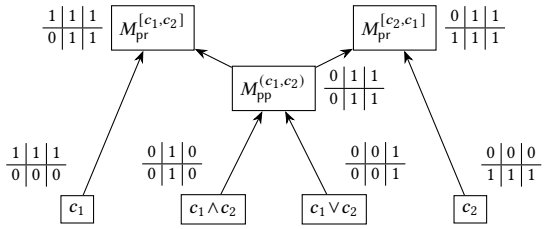


Figure 3: One and two credential mechanisms

priority mechanism with $c_p \sqsupset c_m \sqsupset c_{id}$ (§5.3, App. B). Note that the decision to set $c_p \sqsupset c_m$ is based on the observation that passwords are less likely to be stolen compared to mobile credentials.

In total, the improved mechanism succeeds in 28 scenarios. Notably, it secures an additional scenario among the likely scenarios shown in Tab. 2: (c_p safe, c_m stolen, c_{id} safe). This improvement primarily addresses HDFC's excessive dependence on mobile credentials. However, it introduces a minor inconvenience by extending the time needed for password resets (note that withdrawals already faced delays).

Lastly, we suggest another improvement to the manner in which security notifications are treated by email and messaging providers. Even a short temporary access to a user's mobile device is sufficient for an attacker to delete any sensitive notifications (email or text). We propose to make such notifications *sticky*, i.e., ensure they cannot be deleted for a short period. This improves the synchronous channel's effectiveness significantly.

7.3 Prior Two-Credential Mechanisms

We compare (Fig. 3) all distinct n-credential mechanisms where $n \leq 2$ (that we know of) in a single graph, covering common approaches used in practice, mechanisms from recent prior work, and our new ones. Each rectangle represents a mechanism. Arrows between rectangles show the $\succ$ relation, i.e., an arrow from mechanism M_1 to M_2 signifies $M_2 \succ M_1$. Adjoining a node is the profile of the mechanism restricted to with-safe scenarios. The first row corresponds to c_1 being safe and c_2 being stolen, leaked or lost in that order, and the reverse for the second row.

The bottom row in the figure shows standard approaches to storing a private key: either you store it as is, or split the key ($c_1 \wedge c_2$), or keep two separate copies ($c_1 \vee c_2$).

The middle row represents a Paralysis Proofs [43], which is the most secure 2-credential mechanism from prior work to the best of our knowledge. We model their mechanism, denoted by $M_{pp}^{(c_1,c_2)}$, in App. F. With two credentials, they use an AND-mechanism $c_1 \wedge c_2$ with an additional feature. Any credential can be challenged, and if no response is received within a fixed duration, then the challenged credential is removed. For example, if c_1 is successfully challenged, the new mechanism becomes just c_2. In this manner, they handle credential loss, and achieve better security than both AND and OR mechanisms. But they do not encode the priority between credentials, so it fails in both the scenarios where one credential is safe and another is stolen.

Finally, the top row contains our two maximal mechanisms, namely, $M_{pr}^{[c_1,c_2]}$ and its isomorphism. As Fig. 3 illustrates, our 2-credential mechanisms achieve better security.

8 EXTENSIONS

Our work represents one of the first formal treatments of the authentication problem. We now provide a brief overview of some alternative models and highlight questions for future work.

Partial knowledge: So far we assumed the user had *full knowledge* of the scenario σ. We now explore an extension—where only the attacker fully knows σ, while the user's knowledge is limited only to its own credentials. This extension captures a common real-world setting where the user may not realize whether her credentials are held by an attacker. For instance, if the user possesses a credential, she can only infer that the credential is either *safe* or *leaked*, without being able to ascertain the true state. A similar ambiguity exists between the *lost* and *stolen* states. As the execution progresses, the user may gain additional information about the true state due to e.g., credentials submitted by the attacker. Formally, each credential can now be in one of six (instead of four) states from the user's point of view: {safe,lost,leaked,stolen,{safe,leaked},{lost,stolen}}.

We find that the analysis in our case studies directly applies to the partial knowledge setting. Considering Argent's 2-credential case as an example, the user's winning strategies are very similar to before: revoke a lost-or-stolen credential and assume a safe-or-leaked credential is safe until attacker's submissions reveal otherwise. This happens due to the use of generous timeouts before sensitive actions in Argent (and HDFC), which end up revealing parts of σ to the user, thus making the user's lack of knowledge irrelevant.

However, analyzing the partial knowledge model raises a theoretical question. In brief, it is possible that a winning strategy exists for the user but is impossible to find. This is not possible in our current complete knowledge model because the user can use her knowledge of σ to enumerate attacker strategies and find the winning strategy. In App. G, we show a concrete mechanism that suffers from the above problem. Though the mechanism is theoretical in nature and unlikely to be seen in practice, it does raise an important modeling challenge that we leave for future work to resolve. However, note that our maximal mechanisms remain maximal even in a stronger model where success is defined irrespectively of whether the user can find the winning strategy. The user winning strategies in maximal mechanisms do not suffer from this problem as the simple strategy of submitting all their credentials wins.

Alternate models: Other interesting models to explore include asynchronous communication [40], dynamic scenarios, temporary exclusive knowledge (user knows about a lost credential before the attacker) [17], mechanisms whose logic is hidden from the attacker, and economic incentives [39].

Another crucial direction is to formalize usability considerations. For example, can we design maximally secure mechanisms where the user need not submit too many credentials or wait too long?

9 CONCLUSION

We formalize the authentication problem in a synchronous environment and define the security profile for evaluating authentication mechanisms. After bounding the profile size for any number of credentials n, we discover three types of mechanisms that achieve this bound, and are hence maximally secure. We find that they cover all maximal mechanisms for $n \leq 3$.

Our framework is rich enough to model complex real-world authentication protocols used by millions of users and suggest concrete improvements. Harnessing our results can strengthen existing systems and bolster the security of digital assets and online services.

REFERENCES

[1] [n. d.]. How to get notified on Ethereum. https://vittominacori.medium.com/how-to-get-notified-on-ethereum-or-tokens-received-4b71859a064b.
[2] [n. d.]. Multicall | Uniswap. https://docs.uniswap.org/protocol/reference/periphery/base/Multicall
[3] 2022. Cooling Period: Get time to review newly added beneficiaries. https://www.hdfcbank.com/personal/useful-links/security/security-measures/cooling-period.
[4] 2022. How to change mobile number in HDFC bank: 2 easy ways. https://thebankhelp.com/how-to-change-mobile-number-in-hdfc-bank/.
[5] 2022. How to Update Contact Details at a Branch. https://www.hdfcbank.com/personal/useful-information/change-contact-details.
[6] April 2021. Argent Specification. https://github.com/argentlabs/argent-contracts/blob/develop/specifications/specifications.pdf.
[7] Rajeev Alur and David L. Dill. 1994. A Theory of Timed Automata. *Theor. Comput. Sci.* 126, 2 (1994), 183–235. https://doi.org/10.1016/0304-3975(94)90010-8
[8] Shannon Appelcline. 2021. Using Timelocks to Protect Digital Assets. https://github.com/BlockchainCommons/SmartCustody/blob/master/Docs/Timelocks.md. [Accessed Sep 2022].
[9] Robert J Aumann. 2019. *Lectures on game theory*. CRC Press.
[10] Jean-Philippe Aumasson, Adrian Hamelink, and Omer Shlomovits. 2020. A survey of ECDSA threshold signing. *Cryptology ePrint Archive* (2020).
[11] Fred B. Schneider. 2009, retrieved Oct'22. Authentication for People (draft textbook chapter). https://www.cs.cornell.edu/fbs/publications/chptr.AuthPeople.pdf.
[12] Eduard Banulescu. July 2022. Argent Wallet: Everything You Need To Know. https://beincrypto.com/learn/argent-wallet/.
[13] Praveen Baratam. 2020. Secure Cryptocurrency Exchange & Wallet. https://www.coinvault.tech/wp-content/uploads/2020/10/CoinVault-Secure-Cryptocurrency-Exchange.pdf.
[14] Manuel Barbosa, Alexandra Boldyreva, Shan Chen, and Bogdan Warinschi. 2021. Provable security analysis of FIDO2. In *CRYPTO*. 125–156.
[15] Tal Be'ery. 2018. Threshold Signatures: The Future of Private Keys. https://zengo.com/threshold-signatures-the-future-of-private-keys/.
[16] Matt Bishop. 2004. *Introduction to Computer Security*. Addison-Wesley.
[17] Sam Blackshear, Konstantinos Chalkias, Panagiotis Chatzigiannis, Riyaz Faizullabhoy, Irakliy Khaburzaniya, Eleftherios Kokoris Kogias, Joshua Lind, David Wong, and Tim Zakian. 2021. Reactive Key-Loss Protection in Blockchains. In *WTSC@FC*. 431–450.
[18] Matt Blaze, Joan Feigenbaum, and Angelos D Keromytis. 1998. KeyNote: Trust management for public-key infrastructures. In *Security Protocols*. 59–63.
[19] Matt Blaze, Joan Feigenbaum, and Jack Lacy. 1996. Decentralized trust management. In *IEEE S&P*. 164–173.
[20] 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 *2012 IEEE S&P*. 553–567.
[21] Joseph Bonneau, Andrew Miller, Jeremy Clark, Arvind Narayanan, Joshua A Kroll, and Edward W Felten. 2015. SoK: Research perspectives and challenges for Bitcoin and cryptocurrencies. In *IEEE S&P*. 104–121.
[22] Vitalik Buterin. 2021. Why we need wide adoption of social recovery wallets. https://vitalik.ca/general/2021/01/11/recovery.html.
[23] Chainalysis. 2020. 60% of Bitcoin is Held Long Term as Digital Gold. What About the Rest? https://blog.chainalysis.com/reports/bitcoin-market-data-exchanges-trading/.
[24] Jessica Colnago, Summer Devlin, Maggie Oates, Chelse Swoopes, Lujo Bauer, Lorrie Cranor, and Nicolas Christin. 2018. "It's Not Actually That Horrible": Exploring Adoption of Two-Factor Authentication at a University. In *In ACM CHI*. 1–11.
[25] Ittay Eyal. 2021. On cryptocurrency wallet design. In *Tokenomics*. 4:1–4:16.
[26] Federal Trade Commission. 2022. Consumer sentinel network data book 2021.
[27] Rosario Gennaro, Steven Goldfeder, and Arvind Narayanan. 2016. Threshold-optimal DSA/ECDSA signatures and an application to Bitcoin wallet security. In *ACNS*. 156–174.
[28] Lewis Gudgeon, Pedro Moreno-Sanchez, Stefanie Roos, Patrick McCorry, and Arthur Gervais. 2020. SoK: Layer-two blockchain protocols. In *FC*. 201–226.
[29] Sven Hammann, Saša Radomirović, Ralf Sasse, and David Basin. 2019. User account access graphs. In *ACM CCS*. 1405–1422.
[30] Colin Harper. 2020. Multisignature Wallets Can Keep Your Coins Safer (If You Use Them Right). Coindesk.
[31] Internal Revenue Service. 2022. Taxpayer Guide to Identity Theft. https://www.irs.gov/newsroom/taxpayer-guide-to-identity-theft.
[32] Charlie Jacomme and Steve Kremer. 2021. An extensive formal analysis of multi-factor authentication protocols. *ACM TOPS* 24, 2 (2021), 1–34.
[33] Prashant Jha. 2022. The aftermath of Axie Infinity's $650M Ronin Bridge hack. *Cointelegraph* (2022). https://cointelegraph.com/news/the-aftermath-of-axie-infinity-s-650m-ronin-bridge-hack.
[34] Auguste Kerckhoffs. 1883. La cryptographie militaire. *Journal des sciences militaires* IX (Jan 1883), 5–38.
[35] Auguste Kerckhoffs. 1883. La cryptographie militaire. *Journal des sciences militaires* IX (Feb 1883), 161–191.
[36] Majid Khabbazian, Tejaswi Nadahalli, and Roger Wattenhofer. 2019. Outpost: A responsive lightweight watchtower. In *AFT*. 31–40.
[37] Easwar Vivek Mangipudi, Udit Desai, Mohsen Minaei, Mainack Mondal, and Aniket Kate. 2022. Uncovering Impact of Mental Models towards Adoption of Multi-device Crypto-Wallets. *Cryptology ePrint Archive* (2022).
[38] Patrick McCorry, Surya Bakshi, Iddo Bentov, Sarah Meiklejohn, and Andrew Miller. 2019. Pisa: Arbitration outsourcing for state channels. In *AFT*. 16–30.
[39] Malte Möser, Ittay Eyal, and Emin Gün Sirer. 2016. Bitcoin covenants. In *FC*.
[40] Marwa Mouallem and Ittay Eyal. 2023. Asynchronous Authentication. *arXiv preprint arXiv:2312.13967* (2023).
[41] Lawrence O'Gorman. 2003. Comparing passwords, tokens, and biometrics for user authentication. *Proc. IEEE* 91, 12 (2003), 2021–2040.
[42] Ronald L Rivest and Butler Lampson. 1996. SDSI - A simple distributed security infrastructure. In *USENIX Security*.
[43] Fan Zhang, Philip Daian, Iddo Bentov, Ian Miers, and Ari Juels. 2019. Paralysis Proofs: Secure Dynamic Access Structures for Cryptocurrency Custody and More. In *AFT*. 1–15.

A PRIORITY MECHANISM SMART CONTRACT

A proof-of-concept implementation of the priority mechanism is below. Note that this is strictly an academic prototype meant to elucidate its inner workings, not to be used in production environments.

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.7.0 <0.9.0;

contract PriorityWallet {
    uint public numCredentials;
    mapping(address => uint) public priority;
    bool public withdrawalInProgress = false;
    uint public maxClaimID = 0;
    struct Details {
        bool[] supporters;
        uint amount;
        address payable addr;
    }
    Details[] public claimDetails;
    uint64 constant public delay = 30;
    uint64 public expiryTime;

    // Set the guardians and the priority
        vector. Ideally, you'd also want to deposit some money.
    constructor(address[] memory credentialList_) payable {
        numCredentials = credentialList_.length;
        for (uint i = 0; i < credentialList_.length; i++) {
            priority[credentialList_[i]] = i + 1;
        }
    }
    function getBalance() public view returns (uint) {
        return address(this).balance;
    }
    // start a new withdrawal. Internally creates a new claim
    function initiateWithdrawal() public {
        assert (priority[msg.sender] > 0);
        assert (!withdrawalInProgress);
        withdrawalInProgress = true;
        expiryTime = uint64(block.timestamp + delay);
        // purge previous claim data (if any)
        delete claimDetails;
        maxClaimID = 0;
    }
    function getClaimSupporters
          (uint claimID) public view returns (bool[] memory) {
```

```
        return claimDetails[claimID].supporters;
    }
    // adds approval to an existing claim
    function addApproval(uint claimID) public {
        assert (withdrawalInProgress);
        assert (claimID < maxClaimID);
        assert (uint64(block.timestamp) <= expiryTime);
        assert (priority[msg.sender] > 0);
        claimDetails
             [claimID].supporters[priority[msg.sender]] = true;
    }
    function createNewClaimForWithdrawal(uint amount, address
          payable ToAddress) public returns (uint claimID) {
        assert (priority[msg.sender] > 0);
        assert (withdrawalInProgress
             ); // otherwise call initiateWithdrawal
        assert (uint64(block.timestamp) <= expiryTime);

        bool[] memory supporters = new bool[](numCredentials + 1);
        supporters[priority[msg.sender]] = true;
        claimDetails.push(Details(supporters, amount, ToAddress));

        claimID = maxClaimID; // create new claimID
        maxClaimID = maxClaimID + 1;
    }
    function withdraw() public {
        assert(uint64(block.timestamp) > expiryTime);
        assert(withdrawalInProgress);
        bool[] memory potentialWinners = new bool[](maxClaimID);
        for (uint c = 0; c < maxClaimID; c++) {
            potentialWinners[c] = true;
        }
        for (uint p = 1; p <= numCredentials; p++) {
            bool foundAny = false;
            for (uint c = 0; c < maxClaimID; c++) {
                if (potentialWinners
                     [c] && claimDetails[c].supporters[p]) {
                    foundAny = true;
                }
            }
            if (foundAny) {
                for (uint c = 0; c < maxClaimID; c++) {
                    if (potentialWinners
                         [c] && !claimDetails[c].supporters[p]) {
                        potentialWinners[c] = false;
                    }
                }
            }
        }
        for (uint c = 0; c < maxClaimID; c++) {
            if (potentialWinners[c]) {
                uint winningClaimID = c;
                bool sent = claimDetails[winningClaimID].addr
                     .send(claimDetails[winningClaimID].amount);
                require(sent, "Failed to send Ether");
                withdrawalInProgress = false;
                break;
            }
        }
    }
}
```

B MULTI-TIMER PRIORITY MECHANISMS

In this section, we present a few different approaches to construct priority mechanisms. Unlike the methods specified in Section 5, the below ones are not bounded-delay mechanisms (which employ a single timer) and they employ multiple timers.

Given n credentials $C_{\text{all}} = \{c_1, c_2, ..., c_n\}$ and a priority vector V which is a permutation over the set C_{all}. Let $V = [c_1, c_2, ..., c_n]$ without loss of generality.

Mechanism: Define a sequence of timeouts $x_1 < x_2 < ... < x_n$. The mechanism operates on a simple premise: any participant, denoted as p, may submit a credential c_i at any time to initiate its corresponding timer, t_i. This timer counts towards a predefined threshold, x_i. Once the time elapsed exceeds x_i (i.e., $t_i > x_i$), the mechanism transitions to its final state, resulting in a win for participant p. It's important to note that before a timer t_i surpasses its threshold x_i, it can be reset by resubmitting the corresponding credential c_i.

This setup intricately embeds a priority mechanism through the staggered timing thresholds, ensuring that credentials earlier in the priority vector reach final states sooner.

Profile analysis: If c_1 is safe, then because $\forall j \neq 1,\ x_1 < x_j$, the mechanism succeeds. Similarly if c_1 is stolen, the mechanism fails.

If c_1 is lost, then the mechanism can be recursively analyzed.

If c_1 is leaked, then one party can initiate the timer t_1 but the other party can always cancel it just before the intended time x_1 elapses. Thus mechanism's success depends on the state of c_2, which can be recursively analyzed.

Thus the profile of this mechanism is the same as that of priority mechanism, i.e., it is equivalent to the priority mechanism.

Intuitively, the original priority mechanism only had one timer and it encoded the priorities between all credentials explicitly within a judging function. On the other hand, the above mechanism has n timers and it encodes the priorities indirectly via the n timeouts.

It is also possible to consider hybrids between the two extremes, e.g., employ a k-credential single-timer priority mechanism between credentials $c_1, ..., c_k$ with a time out of x_0 together with a $n-k$-credential multi-timer priority mechanism with time outs $x_{n-k} < x_{n-k+1} < ... < x_n$. Crucially, set $x_0 < x_{n-k}$ which encodes the fact that credentials 1 through k are of higher priority than the later ones.

For example, to instantiate a 3-credential priority mechanism, we can employ a (i) 2-credential single-timer priority mechanism between c_1 and c_2 with a time out of x and (ii) a 1-credential multi-timer priority mechanism, i.e., any party can submit c_3 to start a timer t_3 with a time out of x' such that $x < x'$. In fact, this is the mechanism we recommend for HDFC (§7.2).

C BOUNDED-DELAY AUTOMATON CONSTRUCTION

We briefly explain the construction of a bounded-delay mechanism from a judging function. Given a judging function J defined over a set of credentials C_{all}, we describe an authentication mechanism $M = (C_{\text{all}}, \mathcal{T}, clock, \mathcal{D}, \mathcal{T}_0^{\text{fin}}, \mathcal{T}_1^{\text{fin}})$. Note that we assume that each player submits all their credentials at once to simplify the construction.

Let the set of all AND-credential-guards that use $\wedge$ connector only be G. $|G| = 2^n - 1$ as each credential can either be present or absent and we omit the ε-transition.

The set of all states $\mathcal{T}$ consists of $2 \cdot (2^n - 1)$ intermediate states and 2 final states. An intermediate state t is created for each player ID guard $g^{plr} \in \mathcal{P}$ and credential guard $g_\gamma^{cd} \in G$, with a transition between the start state and this new state, $(I_{\text{init}}, g^{plr}, g_\gamma^{cd}, \perp, \mathsf{True}, t)$, i.e., no clock guard, with clock reset.

The two final states are f_0, f_1. The set $\mathcal{T}_0^{\text{fin}}$ contains f_0 and the set $\mathcal{T}_1^{\text{fin}}$ contains f_1.

The set of all transitions $\mathcal{D}$ consists of edges between the start and intermediate states (explained before) and those between the intermediate and final states (explained next).

There are two different types of transitions between the intermediate and final states. The first type allows the first-mover to win but only after some time elapses. For each intermediate state t, if the player identifier that submitted credentials before is $\mathsf{id}_\gamma \in \mathcal{P}$, then there is a transition $(t,p,\bot,t=l,\mathsf{False},f_\gamma)$, i.e., no credential guard.

The second type allows the other party to win, but only if they submit better credentials. Let C_γ denote the set of credentials submitted by the player id_γ (in g_γ^{cd}) to reach an intermediate state t. Let $p' = \mathsf{id}_{1-\gamma}$.

Find the set of credential guards G' that result in the second-mover winning, i.e., if $\gamma = 0$, find all $g_1^{cd} \in \mathcal{G}_c$ such that C_1 contains the credentials in g_1^{cd} and $J(C_0,C_1) = 1$. And if $p = 1$, find g_1^{cd} s.t. $J(C_1,C_0) = 0$. Add all such guards to G'. For each $g^{cd} \in G'$, there is a transition $(t,p',g^{cd},t<l,\mathsf{False},f_{1-\gamma})$.

D OTHER RESULTS

D.1 Well-formedness proofs

Lemma 10. *Given a permutation V over elements of the set C_{all}, the priority judging function J_{pr}^V is well-formed.*

Proof. We prove each of the three properties (Definition 12). IA requires $(J_{\mathsf{pr}}^V(C_0,C_1)=0) \Leftrightarrow (J_{\mathsf{pr}}^V(C_1,C_0)=1)$. This holds because the priority function selects the unique high-priority credential irrespective of the order. KR requires $C_0 \subset C_1 \implies C_0 \prec C_1$. This holds because C_1 has at least one credential not in C_0 but C_0 has no credentials not in C_1.

To prove TKR, we need to show that given three distinct credential sets C_0, C_1 and C_2, if $C_0 \succ C_1$ and $C_1 \succ C_2$ then $C_0 \not\subseteq C_2$. By contradiction, assume $C_0 \subset C_2$ (since $C_0 \neq C_2$ by definition).

$C_0 \succ C_1$ implies the existence of a credential c that satisfies $c \in C_0 \setminus C_1$ such that $\{c\} \succ C_1 \setminus C_0$.

Observe that $C_0 \subset C_2$ implies $C_1 \setminus C_2 \subseteq C_1 \setminus C_0$.

We can rewrite the previous equation as $\{c\} \succ C_1 \setminus C_2$ due to the transitive nature of the priority judging function.

But the existence of $c \in C_0 \subset C_2$ such that $\{c\} \succ C_1 \setminus C_2$ implies that $C_2 \succ C_1$. This contradicts the TKR assumption $C_1 \succ C_2$, so $C_0 \subset C_2$ cannot be true and TKR is satisfied. □

Lemma 11. *Given a permutation V over set C_{all}, the priority with exception judging function J_{pre}^V is well-formed.*

Proof. It is straightforward to see that the judging function satisfies IA and KR. To satisfy TKR, we want to show that given three different credential sets C_0, C_1 and C_2, if $C_0 \succ C_1$ and $C_1 \succ C_2$ then $C_0 \not\subseteq C_2$.

Since $C_0 \succ C_1$, we have two cases: (A) $C_0 \succ C_1$ is not the exception or (B) $C_0 \succ C_1$ is the exception. Similarly $C_1 \succ C_2$ means: (I) $C_1 \succ C_2$ is not the exception or (II) $C_1 \succ C_2$ is the exception. One of the four cases: A-I, B-I, A-II and B-II must be true. We consider each one separately.

The proof for the case A-I (no exceptions) is exactly the same as the one in Lemma 10.

Case B-II (both exceptions) is impossible because there is only one exception and the three credential sets are different.

The case B-I where $C_0 \succ C_1$ is the exception can also be ruled out due to the way we define an exception, namely, that the credential declared worse by the exception is the worst non-empty set of credentials. For example, if the priority vector is $V = [c_2,c_3,c_1]$, then case B-I corresponds to $C_0 = \{c_1\}$ and $C_1 = \{c_3\}$. But no non-empty set C_2 exists s.t. $C_1 \succ C_2$. So $C_2 = \phi$ and therefore $C_0 \not\subseteq C_2$.

The remaining case is A-II where $C_1 \succ C_2$ is the exception. We prove this by contradiction, i.e., assume $C_0 \subseteq C_2$ or to be precise $C_0 \subset C_2$ because $C_0 \neq C_2$. But no non-empty C_0 exists because the only possible C_0 satisfying $C_0 \subset C_2$ is $C_0 = \phi$ and it does not satisfy $C_0 \succ C_1$. This concludes the proof since we ruled out all the four cases. □

Lemma 12. *Given any ID-agnostic tie-breaking function T, the majority judging function J_{maj}^T is well-formed.*

Proof. We prove the three properties. The IA and KR proofs are immediate. We now prove TKR. Given three different credential sets C_0, C_1 and C_2 s.t. $C_0 \succ C_1$, and $C_1 \succ C_2$, we want to show that $C_0 \not\subseteq C_2$ to satisfy TKR.

For any majority-based judging function, if $C \succ C'$ then $|C| \geq |C'|$. So we have $|C_0| \geq |C_1| \geq |C_2|$.

So either $|C_0| > |C_2|$ or $|C_0| = |C_2|$ is true. If $|C_0| > |C_2|$ then $C_0 \not\subseteq C_2$ and we are done.

It remains to prove for $|C_0| = |C_2|$. But in this case $C_0 \not\subseteq C_2$ follows as $C_0 \neq C_2$ (by definition) and $|C_0| = |C_2|$. □

D.2 Zermelo's

Lemma 13. *Given any mechanism M and scenario σ, either the user has a winning strategy or the attacker does.*

Proof. Like Zermelo [9], we prove the result for a family of games including the current one. Each game has: (1) a starting position determined by a system state, i.e., an automaton state and clock state, and (2) a positive integer m such that if the run does not end in m moves, then the attacker wins. (We will prove at the end that the actual execution we care about is a part of this family.)

We prove by induction on m. If $m = 1$, then any run that ends in more than one move leads to a win for the attacker. If neither party has a valid move to make in the current state, then the attacker wins. Now say the attacker has a valid move. Then irrespective of what the user does, the attacker can order its move at the front and win. This is because either the move is winning and the attacker wins immediately, or if the move is not winning, then the run length is guaranteed to be greater than 1. Now say the attacker doesn't have a valid move, but the user does. Then if that move is winning the user wins. Else the attacker wins because either the run never ends or the run length is greater than 1.

Assume the statement is true for all $m < l$ and prove for $m = l$. Say that the system state is e. By induction hypothesis, if a move is made, either the user or attacker has a winning strategy in the resultant position. If $\exists S^U, Y^{\mathrm{U}} \leftarrow S^U(e)$ s.t. $\forall S^A$ the final set of messages input to the automaton $S^A(e,Y^{\mathrm{U}})$ results in states where the user has a winning strategy, then the user wins. If no such S^U exists, the attacker wins simply by not making a move.

This completes the proof that in all games in the family of games, either the user wins or the attacker does.

We now prove that the original execution is a part of this family. It satisfies the first requirement because the starting state I_{init} is a valid position.

We now show that it satisfies the second requirement. First note that only a finite number of system states are possible (recall that each system state is a tuple $e=(s,t)$). This is because: (a) the number of states in the automaton is finite, and (b) the clocks can be removed to construct a DFA [7].

Next observe that if the same system state repeats in a run, then the attacker wins. This is because both players have deterministic strategies, so the run will be stuck in a loop, i.e., will never end; therefore the attacker wins.

A corollary of the previous two observations is that there exists a value N such that if the length of the run exceeds N, then the attacker wins.

Now the proof is complete because we have seen that for all m there is a winning strategy, in particular for $m \geq N$. □

D.3 Proofs of §5

Lemma 14. *The profile matrix of $M_{\text{pr}}^{[c_1,c_2]}$ is as per Fig. 1.*

Proof. The 3x3 grid with no safe scenarios are unsuccessful, i.e., won by the attacker due to Lemma 3.

Since the priority judging function is well-formed, $M_{\text{pr}}^{[c_1,c_2]}$ is maximal (Lemma 10). Corollary 1 says that all maximal mechanisms are secure in with-safe-no-stolen scenarios $\Sigma_{\text{wsnt}} = \Sigma_{\text{ws}} \cap \Sigma_{\text{nt}}$ and half of the with-safe-with-stolen scenarios ($\Sigma_{\text{wswst}} = \Sigma_{\text{ws}} \cap \Sigma_{\text{wt}}$).

This fixes the values in all scenarios of the matrix except two: the scenario $\sigma = \{\text{safe},\text{stolen}\}$ and its complement $\overline{\sigma} = \{\text{stolen},\text{safe}\}$. Corollary 1 says that at most one of σ, $\overline{\sigma}$ is secure; we need to find out which. The judging function $J_{\text{pr}}^{[c_1,c_2]}$ favors the user in the scenario σ and the attacker in $\overline{\sigma}$. □

Finally, we repeat Lemma 6 and Lemma 7 for the sake of completeness and prove them both now.

Lemma 15. *Given a well-formed mechanism M, let the winning strategy of an attacker in a with-safe-with-stolen scenario $\sigma \in \Sigma_{\text{wswst}}$ be $S^A \neq S^A_{\text{all}}$, then S^A_{all} is also a winning strategy. That is, if $X=(M,\sigma)$ and $\forall S^U : Win_X(S^U,S^A) = A$ then $\forall S^U : Win_X(S^U,S^A_{\text{all}}) = A$.*

Proof. Assume for contradiction the existence of a user strategy S^U such that the strategy S^A_{all} never wins an execution. By Lemma 4, it must be that the user submits a credential set $C^U \subseteq C^U_\sigma$ that is better than or equal to C^A_σ, i.e., $C^U \succeq C^A_\sigma$. Furthermore, the inequality is strict, i.e., $C^U \succ C^A_\sigma$, because if $C^U = C^A_\sigma$ then since $C^U \subseteq C^U_\sigma$, we get $C^A_\sigma \subseteq C^U_\sigma$, which is not true for a with-safe-with-stolen scenario $\sigma \in \Sigma_{\text{wswst}}$.

Start again from the assumption. Since S^A is a successful attacker strategy, it must win an execution against the submit-early strategy with credential set C^U. And because of Lemma 4, the attacker must have submitted a credential set C^A such that $C^A \succeq C^U$.

We have two cases: $C^A = C^U$ or $C^A \succ C^U$. If $C^A = C^U$, since $C^U \succ C^A_\sigma$, we have $C^A \succ C^A_\sigma$. But KR says that if $C^A \subseteq C^A_\sigma$, then $C^A \preceq C^A_\sigma$, which leads to a contradiction.

Next, if $C^A \succ C^U$, since $C^U \succ C^A_\sigma$, the TKR property implies that $C^A \not\subseteq C^A_\sigma$ (note that $C^A \neq C^U$ and $C^U \neq C^A_\sigma$ hold, allowing the use of TKR.) But we have a contradiction as $C^A \subseteq C^A_\sigma$ by definition. So we conclude that the strategy S^A_{all} is also winning. □

Lemma 16. *If S^A_{all} is a winning strategy for the attacker in a with-safe-with-stolen scenario $\sigma \in \Sigma_{\text{wswst}}$, then the user strategy $\bar{S}^U_{\text{all}}$ is a winning strategy for the user in the complement with-safe-with-stolen scenario $\overline{\sigma}$.*

Proof. Since S^A_{all} is a winning strategy in scenario σ, it must win against any user strategy, including S^U_{all}. By Lemma 4, since the user follows a submit-early strategy, the attacker must have submitted C such that $C \succeq C^U_\sigma$. Since the attacker also follows a submit-early strategy S^A_{all}, the attacker submits credentials exactly once. Therefore, it must be that $C = C^A_\sigma$ and hence $C^A_\sigma \succeq C^U_\sigma$. The inequality is strict, i.e., $C^A_\sigma \succ C^U_\sigma$, (C1) since $C^A_\sigma \neq C^U_\sigma$ for a with-safe-with-stolen scenario $\sigma \in \Sigma_{\text{wswst}}$ and due to the mechanism's IA property.

Assume for contradiction that $\bar{S}^U_{\text{all}}$ is not a winning strategy for the user in the complement scenario $\overline{\sigma}$. It means that there exists an attacker strategy $\bar{S}^A$ that wins against $\bar{S}^U_{\text{all}}$ in some executions of $\overline{\sigma}$. Applying Lemma 4 again, the attacker must have submitted a set of credentials $C \subseteq C^A_{\overline{\sigma}}$ such that $C \succeq C^U_{\overline{\sigma}}$. Since $\overline{\sigma}$ is also a with-safe-with-stolen scenario and due to the mechanism's IA property, the inequality is strict, i.e., $C \succ C^U_{\overline{\sigma}}$.

We claim that $C^A_{\overline{\sigma}} \succeq C^U_{\overline{\sigma}}$. (C2) This is because, if instead $C^U_{\overline{\sigma}} \succ C^A_{\overline{\sigma}}$, then because $C \succ C^U_{\overline{\sigma}}$ and TKR, we get $C \not\subseteq C^A_{\overline{\sigma}}$, which is false because $C \subseteq C^A_{\overline{\sigma}}$.

By definition, for any pair of complement scenarios, we have $C^U_\sigma = C^A_{\overline{\sigma}}$ and $C^U_{\overline{\sigma}} = C^A_\sigma$. Recasting the equation C2, we get $C^U_\sigma \succeq C^A_\sigma$, which contradicts the equation C1. Thus S^A_{all} is a winning user strategy in $\overline{\sigma}$. □

D.4 Count of majority judging functions

We now count the number of different majority functions, i.e., tie-breaking functions. A tie occurs when $C_0 \neq C_1$ but $|C_0| = |C_1|$. Let $|C_0| = |C_1| = s$ where $1 \leq s < n$ (note that s can't be equal to n since $C_0 \neq C_1$). For a given s, there are $f(n,s) = \frac{\binom{n}{s}(\binom{n}{s}-1)}{2}$, possible tie situations. This is because, there are $\binom{n}{s}$ ways of picking an s-sized set C_0, and $\binom{n}{s}-1$ ways of picking another set $C_1 \neq C_0$. We divide by two as each tie situation repeats twice. Summing over all s, the total number of distinct tie situations is $q(n) = \sum_{s=1}^{n-1} f(n,s) = \binom{2n-1}{n-1} - 2^{n-1}$. A tie-breaking function can decide each situation in two ways: pick C_0 or C_1. So there are $2^{q(n)}$ different tie-breaking functions, or in other words, $2^{q(n)}$ different majority functions.

D.5 Proofs of §4

We now provide the proof of Lemma 2.

Proof. Let S^U denote a winning user strategy in the extended scenario $X = (M,\sigma)$ (one such S^U is the submit-all strategy S^U_{all}). Since S^U wins in all executions, we have $\forall S^A : Win_X(S^U,S^A) = \text{U}$. We now produce a winning strategy $\bar{S}^A$ for the attacker in $\bar{X} = (M,\overline{\sigma})$ based on S^U. To do this, we need to show two things, namely, that S^U is a playable strategy for the attacker in $\overline{\sigma}$, and that it succeeds in at least one execution.

The first follows in a straightforward way from the definition of a complement. It is easy to see that $C^U_\sigma = C^A_{\overline{\sigma}}$. Therefore, any user strategy, including S^U, in the scenario σ is a playable attacker messaging strategy in the complement scenario $\overline{\sigma}$.

Next we need to show that the attacker wins an execution with S^U in $\overline{\sigma}$. Recall that Ord^A denotes the attacker-first reordering strategy. Let the attacker's strategy be $\bar{S}^A = (S^U, Ord^A)$. We need to show that no user strategy succeeds in all executions of $\bar{X} = (M, \overline{\sigma})$, i.e., $\nexists \bar{S}^U : Win_{\bar{X}}(\bar{S}^U, \bar{S}^A) = \text{U}$.

We prove by contradiction. Assume the existence of a winning user strategy $\bar{S}^U$, i.e., $Win_{\bar{X}}(\bar{S}^U, \bar{S}^A) = \text{U}$. We will use the successful strategy $\bar{S}^U$ to devise a strategy S^A for the attacker in the original scenario.

Apply Lemma 1 by setting $\sigma_1 = \overline{\sigma}$, $\sigma_2 = \sigma$, $S_1 = \bar{S}^U$ and $S_2 = S^U$. A pre-condition for this lemma is that S_1 ($\bar{S}^U$) needs to be playable by the attacker in σ_2 (σ), which holds because $C^U_{\overline{\sigma}} = C^A_\sigma$. Lemma 1 guarantees the existence of an ordering strategy S^A_{ord} such that $Win_{\bar{X}}(\bar{S}^U, \bar{S}^A) = \text{U} \implies Win_X(S^U, S^A) = \text{A}$. We thus found an attacker strategy (S^A) that wins an execution against the user winning strategy S^U in the original scenario σ—a contradiction. Therefore, the assumption was wrong and there does not exist a winning strategy for $\overline{\sigma}$. □

We now provide the proof of Lemma 3.

Proof. By contradiction, assume there exists a mechanism M that succeeds in a bad scenario, i.e., $Suc(M, \sigma^{\text{bad}})$. Denote the extended scenario $X = (M, \sigma^{\text{bad}})$. Let the winning user strategy be S^U, so for all S^A, $Win_X(S^U, S^A) = \text{U}$. Consider the attacker strategy $S^*_1 = (S^U, Ord^A)$. We have $Win_X(S^U, S^*_1) = \text{U}$.

Apply Lemma 1 by setting $\sigma_1 = \sigma$, $\sigma_2 = \sigma$, $S_1 = S^U$, and $S_2 = S^U$. A pre-condition for this lemma is that S^U is playable by both user and attacker in σ. This is true because in any scenario $\sigma \in \sigma^{\text{bad}}$, $C^U_\sigma \subseteq C^A_\sigma$. Consequently, for any set of credentials C, if $C \subseteq C^U_\sigma$ then $C \subseteq C^A_\sigma$. So S^U is playable by the attacker.

We have $Win_X(S^U, S^*_1) = \text{U}$, therefore due to Lemma 1, there exists some S^A_{ord} such that if $S^*_2 = (S^U, S^A_{\text{ord}})$ then $Win_X(S^U, S^*_2) = \text{A}$. This completes the proof because S^*_2 wins against S^U, contradicting the assumption that S^U is a winning user strategy. Therefore the assumption was wrong and no user strategy in a bad scenario is winning. □

D.6 Complete Maximal Sets Proofs

Lemma 17. *For all n, the size of all n-credential complete maximal sets is the same.*

Proof. Consider two different complete maximal sets O_1 and O_2. Each mechanism $M_2 \in O_2$ must be equivalent to or worse from a mechanism $M_1 \in O_1$ (Definition 14). But all mechanisms in a complete maximal set are maximal, therefore it must be that $M_2 \cong M_1$. So, for each mechanism in O_2, there exists an equivalent mechanism in the other set O_1, and vice versa. And because no two mechanisms within O_1 (or O_2) can be equivalent, there exists a one-to-one mapping between O_1 and O_2. So $|O_1| = |O_2|$. □

We now formalize settings and prove some results on Complete Maximal Sets.

Definition 15. *Given a credential set C_{all}, a setting P specifies the probability distributions governing the credential states for all credentials [25]. Let $P(c,\ state)$ denotes the probability that the credential c is in the given state. We have, $\forall c \in C_{\text{all}}$, $P(c, \textit{safe}) + P(c, \textit{stolen}) + P(c, \textit{leaked}) + P(c, \textit{lost}) = 1$.*

For example, with two credentials, if c_1 is always safe and c_2 is either safe or stolen with equal probability, we can represent it through a setting as follows: $P(c_1, \text{safe}) = 1$, $P(c_2, \text{safe}) = 0.5$, and $P(c_2, \text{stolen}) = 0.5$.

Given a setting P, the probability that a scenario σ occurs is given by $P_\sigma = \Pi_{i \in 1,2,\ldots,n} P(c_i, \sigma_i)$. Further, given a mechanism M, its success probability in a setting is given by $\Gamma(M,P) = \Sigma_{\sigma \in prof(M)} P_\sigma$.

We find that one of the complete maximal set mechanisms is the most secure (succeeds the most) in any given setting, i.e., $\exists M^* \in O_n, \Gamma(M^*, P) \geq \Gamma(M, P)$ for any M (using the same credential set). The proof follows from the definition of complete sets and is in App. D.

We now prove that a mechanism in the complete maximal set is the most secure in any given setting.

Lemma 18. *Given a set of n credentials C_{all} and an associated setting P, one of the mechanisms in the complete maximal set is the most secure in this setting, i.e., $\exists M^* \in O_n$ such that for all mechanisms M using C_{all}, we have that $\Gamma(M,P) \leq \Gamma(M^*,P)$.*

Proof. By contradiction, assume the existence of a mechanism M' using C_{all} such that it is not part of the complete set, i.e., $M' \notin O_n$, and is more secure than any complete set mechanism $\Gamma(M',P) > \Gamma(M'',P)$ for all $M'' \in O_n$.

By Definition 14, given any n-credential mechanism M, there exists a mechanism $M'' \in O_n$ such that $M \leq M''$. Applying it for M', denote the corresponding mechanism in the complete set by M''. We have $M' \leq M''$, or equivalently $prof(M') \subseteq prof(M'')$.

It follows that $\Gamma(M',\ P) = \Sigma_{\sigma \in prof(M')} P_\sigma \leq \Gamma(M'',\ P) = \Sigma_{\sigma \in prof(M'')}$. So we arrive at a contradiction. Therefore, it must be that one of the complete set mechanisms is the most secure in any setting. □

Note that the above lemma doesn't rule out the possibility that a non-maximal mechanism is as secure as a maximal one in a given setting. For example, given two credentials, consider the setting: $P(c_1, \text{safe}) = 1$, $P(c_2, \text{safe}) = 0.5$, $P(c_2, \text{stolen}) = 0.5$. In this setting, the mechanism that just uses c_1 without using c_2 would be as secure as our 2-credential priority mechanism (And you might indeed want to use the simpler one-credential mechanism in practice for usability reasons).

E COMPLETE SETS DETAILS

E.1 $O_{\text{maj},3}$ mechanisms

The twelve majority mechanisms in O_3 are:

Both linear (6 mechanisms): Fix $[c_1, c_2, c_3]$ as the linear priority rule to break ties between 1-credential sets, and use all the six permutations of $[c_1, c_2, c_3]$ to break ties between 2-credential sets.

Both cyclic (2 mechanisms): Two possible cyclic rules exist: clockwise ($c_1 \succ c_2, c_2 \succ c_3, c_3 \succ c_1$) or counter-clockwise. The two mechanisms are: use same or different rules for both 1-credential and 2-credential tie-breakers.

Linear and cyclic (4 mechanisms): Fix $[c_1, c_2, c_3]$ as the linear priority rule to break ties between 1-credential sets and use clockwise or counter-clockwise rule for 2-credential sets to produce 2 mechanisms. Use cyclic rule for 1-credential sets and linear rule for 2-credential sets to obtain two more.

E.2 Proving 3-credential complete sets

Theorem 3. *A complete maximal set of 3-credential mechanisms is* $O_3 = O_{\text{pr},3} \cup O_{\text{maj},3}$.

Proving the first two requirements of Definition 14 is straightforward. We prove the last requirement that no mechanism is better than a mechanism in O_3 by contradiction. Assume that a mechanism M exists such that it is incomparable with any mechanism in O_3. We encode this as a constraint in a constraint solver and have it search for a satisfying profile. We also add another constraint relying on the observation that if M fails in a scenario σ, then it must fail in all scenarios $\widehat{\sigma}$ that are worse than σ where $\widehat{\sigma}$ is *worse* than σ if the attacker knows the same or more credentials whereas the user knows the same or fewer credentials. We find that the constraint solver is unable to find a solution, therefore no such mechanism exists. The details are in App. E.

Note 2. *It remains an open question how to analytically find complete maximal sets for larger number of credentials.*

Before proving that these mechanisms form the complete maximal set for 3 credentials we present a few lemmas. The first one says that if both players use fixed strategies, then a change in the scenario does not change the execution winner.

Lemma 19. *Given two scenarios* σ, σ' *using the same mechanism* M *and a user strategy* S^U*, attacker strategy* S^A *such that they can be employed in both the scenarios. Then* $Win_\sigma(S^U, S^A) = Win_{\sigma'}(S^U, S^A)$.

Proof. The proof follows straightforwardly because we are using a deterministic automaton and the strategies are deterministic, so irrespective of the scenario, the executions will be the same, and hence the winner. □

The next lemma says that, if the mechanism fails in a scenario σ, then it also fails in all scenarios $\widehat{\sigma}$ that are worse than σ. $\widehat{\sigma}$ is *worse* than σ as the attacker knows (equal or) more credentials whereas the user knows (equal or) fewer credentials. Formally, given a scenario σ, we define a worse scenario $\widehat{\sigma}$ through the following transform:

$$\begin{aligned} \mathsf{safe} &\mapsto \mathsf{safe/leaked/lost/stolen} \\ \mathsf{stolen} &\mapsto \mathsf{stolen} \\ \mathsf{leaked} &\mapsto \mathsf{leaked/stolen} \\ \mathsf{lost} &\mapsto \mathsf{lost/stolen}. \end{aligned}$$

Lemma 20. *If a mechanism fails in a scenario* σ*, then it also fails in a worse scenario* $\widehat{\sigma}$*,* $\forall M, \neg Suc(M,\sigma) \implies \neg Suc(M,\widehat{\sigma})$.

Note that proving the above also implies that success in a worse scenario $\widehat{\sigma}$ implies success in σ.

Proof. Observe that (a) $C^A_\sigma \subseteq C^A_{\widehat{\sigma}}$ and (b) $C^U_{\widehat{\sigma}} \subseteq C^U_\sigma$.

We prove by contradiction. That is, assume that the attacker succeeds in a scenario σ but the user succeeds in the transformed scenario $\widehat{\sigma}$. Let the successful strategy for the user in $\widehat{\sigma}$ be S^U. Since $C^U_{\widehat{\sigma}} \subseteq C^U_\sigma$, the user can employ the strategy S^U in the scenario σ. By our initial assumption, the attacker succeeds in σ. Let the successful attacker strategy be S^A. S^A must win some executions against any user strategy including S^U. So we have $Win_\sigma(S^U, S^A) = \text{A}$.

Since $C^A_\sigma \subseteq C^A_{\widehat{\sigma}}$, the attacker can employ the strategy S^A in the scenario $\widehat{\sigma}$. Any execution in the scenario $\widehat{\sigma}$ where the user employs S^U and the attacker employs S^A results in a win for the user. So we have $Win_{\widehat{\sigma}}(S^U, S^A) = \text{U}$.

But σ and $\widehat{\sigma}$ use the same underlying mechanism M. And if the strategies of the two parties are same, then both $Win_\sigma(S^U, S^A) = \text{A}$ and $Win_{\widehat{\sigma}}(S^U, S^A) = \text{U}$ cannot be true due to Lemma 19 and we arrive at a contradiction. □

We now prove Theorem 3, i.e., that O_3 is made up of the two priority and twelve majority mechanisms only.

Proof. Proving a complete maximal set requires satisfying three requirements per Definition 14:

(1) All mechanisms in O_3 must be maximal.
(2) All mechanisms in O_3 must be incomparable.
(3) No other mechanism exists s.t. it is incomparable to each mechanism in O_3.

The first requirement is met due to Lemma 10, Lemma 11 and Lemma 12. The second one is also met, as can be verified by inspecting the 76 mechanism profiles. (We cross-check this through code.)

We prove the third requirement using a similar approach taken in Theorem 2, but done at a larger scale. We use a constraint solver to show that no 3-credential mechanism M exists satisfying two types of constraints, explained below.

The first type of constraint encodes that M must be incomparable with any mechanism in O_3. Satisfying it requires that for each mechanism $M' \in O_3$, there exists at least one scenario $\sigma \in prof(M)$ that satisfies $\sigma \notin prof(M')$. This scenario σ must be a with-safe scenario due to Lemma 3. Rephrasing the above, we get $\exists \sigma \in prof(M)$ such that $\sigma \in \Sigma_{\text{ws}} \setminus prof(M')$.

Observe that for any $M' \in O_3$, $prof(M')$ contains all the with-safe scenarios except some with-safe-with-stolen scenarios. Therefore $\Sigma_{\text{ws}} \setminus prof(M') \subset \Sigma_{\text{wswst}}$. And consequently, $prof(M)$ must include at least one with-safe-with-stolen scenario.

So to prove that no mechanism M exists, it suffices to prove that $prof(M)$ does not contain a with-safe-with-stolen scenario. Let $prof_{\text{wswt}}(M)$ denote the subset of $prof(M)$ containing just the with-safe-with-stolen scenarios. We want to show that $|prof_{\text{wswt}}(M)| = 0$.

We prove this by consider the set of all subsets of the 18 with-safe-with-stolen scenarios satisfying Lemma 2, i.e., no two scenarios in a subset are complements of each other. Denote this set by Δ ($|\Delta| = 3^9$ because the 18 with-safe-with-stolen scenarios can be arranged into 9 rows with each row containing 2 complement scenarios. And we have three ways of making a selection in each row: {none, first scenario, second scenario}). Observe that this is the set of all possible values for $prof_{\text{wswt}}$, i.e., for any mechanism M, it must be that $prof_{\text{wswt}}(M) \in \Delta$.

And we want to find a set in Δ satisfying the two constraints. The first constraint, explained before, is $\forall M' \in O_3, prof_{\text{wswt}}(M) \cap (\Sigma_{\text{ws}} \setminus prof(M')) \neq \phi$.

The second constraint relies on the following observation. If the mechanism wins in a with-safe-with-stolen scenario σ, Lemma 2 implies that it fails in the complement scenario $\overline{\sigma}$. But

c_1 **safe**

c_2 \ c_3	St	Le	Lo	Sa
St	1	1	1	1
Le	1	1	1	1
Lo	1	1	1	1
Sa	1	1	1	1

c_1 **lost**

c_2 \ c_3	St	Le	Lo	Sa
St	0	0	0	0
Le	0	0	0	1
Lo	0	0	0	1
Sa	1	1	1	1

c_1 **leaked**

c_2 \ c_3	St	Le	Lo	Sa
St	0	0	0	0
Le	0	0	0	1
Lo	0	0	0	1
Sa	1	1	1	1

c_1 **stolen**

c_2 \ c_3	St	Le	Lo	Sa
St	0	0	0	0
Le	0	0	0	0
Lo	0	0	0	0
Sa	0	0	0	0

Table 3: Profile of a priority 3-credential mechanism $M_{\text{pr}}^{[c_1,c_2,c_3]}$

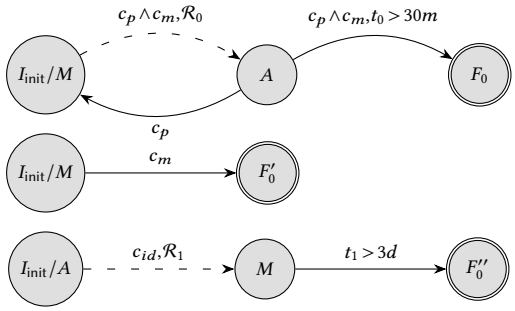


Figure 4: HDFC bank sub-automata: send money to a new account, change password, and change mobile number respectively.

then, due to Lemma 20, the mechanism fails in any scenario worse than $\overline{\sigma}$. For example, $prof(M)$ cannot contain the two scenarios $\sigma = \{\text{stolen},\text{stolen},\text{safe}\}$ and $\sigma' = \{\text{lost},\text{safe},\text{stolen}\}$. This is because the mechanism fails in the complement of σ, $\overline{\sigma} = \{\text{safe},\text{safe},\text{stolen}\}$, and σ' is a worse scenario than $\overline{\sigma}$.

The second constraint is $\forall \sigma \in prof_{\text{wswt}}(M)$, $\nexists \sigma' \in prof_{\text{wswt}}(M)$ s.t. the complement $\overline{\sigma}$ is worse than σ'.

The constraint solver[3] outputs the empty set, i.e., no such $prof_{\text{wswt}}(M)$ exists and hence $|prof_{\text{wswt}}(M)| = 0$. This completes the proof that O_3 is complete. □

F APPLICATIONS

We present details of Argent in App. F.1 and HDFC bank in App. F.2. The Paralysis Proofs [43] mechanism is in Fig. 5.

F.1 Argent

F.1.1 Modeling choice. In the main paper, we modelled both credential changes and money transfers as final states. This is because we find that, in the case of Argent, if the user has a winning strategy in

[3]Code at https://pastebin.com/RqjesNZe for anonymous submission.

c_{id} **safe**

c_m \ c_p	St	Le	Lo	Sa
St	0	0	0	0
Le	0	0	0	0
Lo	1	1	1	1
Sa	1	1	1	1

c_{id} **lost**

c_m \ c_p	St	Le	Lo	Sa
St	0	0	0	0
Le	0	0	0	0
Lo	0	0	0	0
Sa	1	1	1	1

c_{id} **leaked**

c_m \ c_p	St	Le	Lo	Sa
St	0	0	0	0
Le	0	0	0	0
Lo	0	0	0	0
Sa	1	1	1	1

c_{id} **stolen**

c_m \ c_p	St	Le	Lo	Sa
St	0	0	0	0
Le	0	0	0	0
Lo	0	0	0	0
Sa	1	1	1	1

Table 4: Profile of the bank mechanism.

c_{id} **safe**

c_m \ c_p	St	Le	Lo	Sa
St	0	0	0	1
Le	0	1	1	1
Lo	0	1	1	1
Sa	0	1	1	1

c_{id} **lost**

c_m \ c_p	St	Le	Lo	Sa
St	0	0	0	0
Le	0	0	0	0
Lo	0	0	0	0
Sa	1	1	1	1

c_{id} **leaked**

c_m \ c_p	St	Le	Lo	Sa
St	0	0	0	0
Le	0	0	0	0
Lo	0	0	0	0
Sa	1	1	1	1

c_{id} **stolen**

c_m \ c_p	St	Le	Lo	Sa
St	0	0	0	0
Le	0	0	0	0
Lo	0	0	0	0
Sa	1	1	1	1

Table 5: Profile of the newly proposed bank mechanism.

the above model, i.e., they are are able to reach a final state, then they can also transfer money. In particular, if the Argent mechanism succeeds, then there exists a winning user strategy where each winning execution ends either in a money transfer or in a change of an unsafe credential. This process can be iterated sufficiently many times to remove all the unsafe credentials and transfer money. A similar statement also holds for the attacker, except that they are either able to successfully transfer money or keep the mechanism unusable. Note that the winning strategies (both user and attacker) provided in §7.1.2 for the Argent 1-guardian case already satisfy this property.

Finally, note that the above isn't true in general. There may be algorithms where a credential change does not imply a win. To support such mechanisms, we need to extend our model to capture scenario changes. One could do this by introducing a new type of automaton state that leads to a reconfiguration, i.e., the execution switches to a new automaton and a new scenario. This is only meaningful if scenarios change over time, so a reconfiguration can secure a mechanism at a future time. We leave such analysis, which

c_1 **safe**

c_2 \ c_3	St	Le	Lo	Sa
St	0	0	0	1
Le	0	0	0	1
Lo	0	0	0	1
Sa	1	1	1	1

c_1 **lost**

c_2 \ c_3	St	Le	Lo	Sa
St	0	0	0	0
Le	0	0	0	1
Lo	0	0	0	0
Sa	0	1	0	1

c_1 **leaked**

c_2 \ c_3	St	Le	Lo	Sa
St	0	0	0	0
Le	0	0	0	0
Lo	0	0	0	1
Sa	0	0	1	1

c_1 **stolen**

c_2 \ c_3	St	Le	Lo	Sa
St	0	0	0	0
Le	0	0	0	0
Lo	0	0	0	0
Sa	0	0	0	1

Table 6: Profile of the 2-out-of-3 mechanism.

c_1 **safe**

c_2 \ c_3	St	Le	Lo	Sa
St	0	0	0	0
Le	0	0	0	0
Lo	0	0	0	1
Sa	0	0	1	1

c_1 **lost**

c_2 \ c_3	St	Le	Lo	Sa
St	0	0	0	0
Le	0	0	0	0
Lo	0	0	0	1
Sa	0	0	1	1

c_1 **leaked**

c_2 \ c_3	St	Le	Lo	Sa
St	0	0	0	0
Le	0	0	0	0
Lo	0	0	0	0
Sa	0	0	0	0

c_1 **stolen**

c_2 \ c_3	St	Le	Lo	Sa
St	0	0	0	0
Le	0	0	0	0
Lo	0	0	0	0
Sa	0	0	0	0

Table 7: Profile of the 1-out-of-3 mechanism.

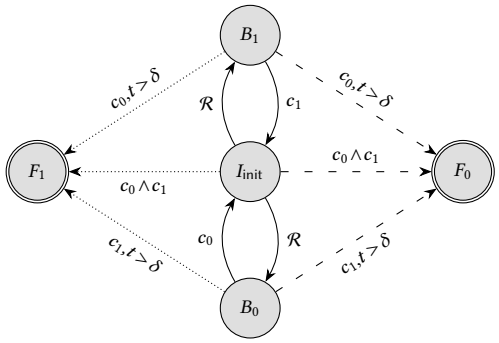


Figure 5: Paralysis Proofs Mechanism [43] with 2-credentials

o **safe**

g_1 \ g_2	St	Le	Lo	Sa
St	1	1	1	1
Le	1	1	1	1
Lo	1	1	1	1
Sa	1	1	1	1

o **lost**

g_1 \ g_2	St	Le	Lo	Sa
St	0	0	0	0
Le	0	0	0	0
Lo	0	0	0	1
Sa	0	0	1	1

o **leaked**

g_1 \ g_2	St	Le	Lo	Sa
St	0	0	0	0
Le	0	0	0	0
Lo	0	0	0	1
Sa	0	0	1	1

o **stolen**

g_1 \ g_2	St	Le	Lo	Sa
St	0	0	0	0
Le	0	0	0	0
Lo	0	0	0	0
Sa	0	0	0	0

Table 8: Profile of Argent with 2 guardians

includes a definition of dynamic scenarios where key availability changes over time, for future work.

F.1.2 2 guardians. We analyze the profile of Argent's mechanism with 2 guardians M^2_{Arg} (Tab. 8). Since any mechanism fails in all no-safe scenarios (Lemma 3), we only discuss with-safe scenarios. Like in the 1-guardian case, at a high level, the user's strategy is to revoke any lost or stolen credential.

o safe / stolen: If the owner is safe, the user can revoke all the leaked and stolen guardians (if any), and win. Note that even if the attacker initiates an owner change before (possible in scenarios where the attacker knows one guardian), the user can delete the unsafe guardians after learning that the guardian is leaked (Fig. 2f). Similarly, if the owner is stolen, the attacker can execute the above strategy and win.

o lost: If each of the guardians is either lost or safe, the user wins by using the safe credential to change the lost credentials. Note that a single guardian is enough to change the owner.

In case one of the guardians gets either leaked or stolen, the attacker wins. The attacker's winning strategy is to make the wallet unusable by bringing the automaton to the owner change state W_o (requires one guardian). Even if the user can cancel the owner change (possible in certain scenarios), the attacker can immediately initiate another owner change.

o leaked: If both guardians are safe or if one is safe and another is lost, the user wins because she is able to initiate an owner change and the attacker does not know enough credentials to cancel it. Note that starting a guardian change does not help the attacker as the user can cancel it just before the 1.5 day period ends.

If one of the guardians is leaked / stolen, the attacker wins by executing a fast withdrawal.

In summary, the profile of M^2_{Arg} has 22 scenarios in comparison to our maximal mechanisms with 28 scenarios. A more commonly used 2-out-of-3 mechanism's profile has 14 scenarios in comparison (see Tab. 6). But M^2_{Arg} is worse than the priority mechanism where

owner credential has the highest priority, i.e., $M^2_{\text{Arg}} \prec M^{[o,g_1,g_2]}_{\text{pr}}$ (see Tab. 3).

Multicalls: With multicalls, the mechanism additionally wins in the scenarios: (o lost, g_1 leaked, g_2 safe) and (o lost, g_1 safe, g_2 leaked). Everything else remains the same.

The winning strategy based on the automaton's state: (a) if it is W_o and the owner change is not started by the user, cancel the current owner change, and immediately start a new owner change, and (b) if the state is anything else, execute an owner change. After the owner becomes safe, we move to an already analyzed scenario where the mechanism succeeds. Observe that $M^2_{\text{Arg}} \prec M^{[o,g_1,g_2]}_{\text{pr}}$ holds even with multicalls.

F.2 Bank

We illustrate the bank automaton in Fig. 4 and explain its resultant profile (Tab. 4).

If c_m is safe, then irrespective of the state of password, the mechanism succeeds because the user can reset password (if it is unsafe) and then send money to a new account. Even if c_{id} is unsafe, the user can send all the money to a new account in the three days it takes to change the mobile number. Note that this is true because we assume that the user is notified of any mobile number change requests made by the attacker.

The mechanism fails if c_m is leaked or stolen (corresponding to phone hijacking attacks like SIM swapping) because the attacker can reset the password.

The mechanism succeeds if c_m is lost and c_{id} is safe because the user can change their mobile number by waiting for 3 days.

Finally, the mechanism fails if c_m is lost and c_{id} is unsafe because either the attacker changes the mobile (c_{id} leaked / stolen) or no one does and the account is unusable (c_{id} lost).

F.2.1 Improving bank's profile. We recommend using a 2-credential maximal mechanism between the password and mobile credentials and leaving the portion involving the ID credential as is. This is because submitting the ID credential involves a demanding task of submitting documents by physcially visiting a bank. Thus, we adopt the philosophy of the HDFC bank automaton to use the ID credential sparingly.

Concretely, we recommend using a 2-credential priority mechanism with a higher priority assigned to c_m over c_p. This mechanism can be used for withdrawals and resetting the password, i.e., the first two sub-automata in Fig. 4 (leaving the third one as is). For usability reasons, we assume that the delay in the priority mechanism is less than that of the third automaton (3d), e.g., say 30m.

Like the original mechanism's profile, if c_m is safe, then irrespective of the state of other credentials, the mechanism succeeds as (a) c_m has higher priority than c_p and (b) it is faster to withdraw money than change c_m (3 days).

Say c_m is lost or leaked. If c_p is safe, then the mechanism succeeds because c_m is known to both but c_p is only known to the user.

Finally, if both c_m and c_p are lost while c_{id} is safe, then the user can show their ID to regain access to c_m after 3 days. Similarly, if (c_m lost, c_p leaked, c_{id} safe), then the same strategy allows the user to regain access (as the user can deter any attempts to withdraw money for the three days).

This mechanism succeeds in 26 scenarios, which is still three less than our maximal mechanisms. In particular, comparing it with the profile of a priority 3-credential mechanism (Tab. 3), we see that the three differing scenarios are (c_{id} safe; c_m leaked; c_p leaked) and (c_{id} safe; c_m leaked; c_p lost). In all three, we are unable to design a mechanism where the mechanism succeeds because we are unable to make the user show the ID credential quickly. One approach to fix this might be to introduce a locked state that gets triggered as soon as the mobile change process is initiated (however it has other implications, e.g., on states where c_{id} is stolen, but that might be acceptable because the likelihood of theft for c_{id} is low).

F.2.2 Analyzing success probabilities. We make some baseline assumptions about the probability distribution P governing the credential states to arrive at the optimal mechanism. Say that the ID credential is always safe, $P(c_{id}, \mathsf{safe}) = 1$; passwords are equally prone to loss and leakage, i.e., $P(c_p,\mathsf{lost}) = P(c_p,\mathsf{leaked}) = 0.15$, $P(c_p,\mathsf{safe}) = 0.7$; and the mobile credential is more safe than a password but is prone to getting stolen (e.g., SIM swaps), i.e., $P(c_m,\mathsf{safe}) = 0.9$, $P(c_m,\mathsf{stolen}) = 0.1$.

In this setting, the original HDFC mechanism's success probability is 0.9 (probability that c_m is safe or lost).

Of our maximal mechanisms, priority mechanisms where the ID credential is prioritized the most are the most secure, i.e., achieve a success probability of 1. But in practice, submitting an ID credential requires physically submitting documents at the bank. So we aim to keep the usage of the ID credential to a minimum, much like the existing HDFC mechanism.

Therefore, we recommend making a small tweak that does not change the HDFC mechanism by much: use a 2-credential priority mechanism between c_m and c_p (with $c_p \sqsupset c_m$) before withdrawals or password reset, keeping rest of the mechanism involving the ID credential same as before. Surprisingly, this small modification makes the mechanism maximally secure – it results in a multi-timer priority mechanism (§5.3, App. B) with $c_p \sqsupset c_m \sqsupset c_{id}$. Concretely, in the above setting, the success probability of our proposed mechanism is 0.97, higher than HDFC's success probability of 0.9. This security improvement comes from winning in an additional scenario: (c_p safe, c_m stolen, c_{id} safe). In essence, our mechanism improves upon HDFC by reducing the over-reliance of it on the security of the mobile credential, c_m. However, it introduces a minor inconvenience by extending the time needed for password resets (note that withdrawals already faced delays).

Note that we set $c_p \sqsupset c_m$ in order to exploit the fact that passwords are less likely to be stolen compared to mobile (the success probability of the mechanism with $c_m \sqsupset c_p \sqsupset c_{id}$ is only 0.9).

G PARTIAL KNOWLEDGE CHALLENGE

Given three credentials 1, 2, and 3, credential 1 beats 2, 2 beats 3, 3 beats 1, and submitting more than one credential loses. Take the two scenarios in the partial knowledge model (§8):

(1) Credentials 1 and 2 are safe, but credential 3 is leaked. Submitting credential 2 is a winning strategy.
(2) Credentials 2 and 3 are safe, but credential 1 is leaked. Submitting credential 3 is a winning strategy.

The scenarios are indistinguishable for the user - all the credentials are safe-or-leaked. And even though there exists a winning user strategy in each scenario, the user cannot determine it.