

Asynchronous Authentication

Marwa Mouallem
marwamouallem@cs.technion.ac.il
Technion
Haifa, Israel

Ittay Eyal
ittay@technion.ac.il
Technion
Haifa, Israel

ABSTRACT

A myriad of authentication mechanisms embody a continuous evolution from verbal passwords in ancient times to contemporary multi-factor authentication: Cryptocurrency wallets advanced from a single signing key to using a handful of well-kept credentials, and for online services, the infamous “security questions” were all but abandoned. Nevertheless, digital asset heists and numerous identity theft cases illustrate the urgent need to revisit the fundamentals of user authentication.

We abstract away credential details and formalize the general, common case of *asynchronous authentication*, with unbounded message propagation time. Given credentials’ fault probabilities (e.g., loss or leak), we seek mechanisms with maximal success probability. Such analysis was not possible before due to the large number of possible mechanisms. We show that every mechanism is dominated by some *Boolean mechanism*—defined by a monotonic Boolean function on presented credentials. We present an algorithm for finding approximately optimal mechanisms by leveraging the problem structure to reduce complexity by orders of magnitude.

The algorithm immediately revealed two surprising results: Accurately incorporating easily-lost credentials improves cryptocurrency wallet security by orders of magnitude. And novel usage of (easily-leaked) security questions improves authentication security for online services.

1 INTRODUCTION

Authentication is a cornerstone of access control and security [54, 57]. From ancient times [17] to the early days of the Internet, passwords were often sufficient [10]. As stakes grow higher with the adoption of online services, companies deploy advanced *authentication mechanisms* [1, 2], and abandon tools like the easily guessable “security questions” like “name of first pet.” Nascent technologies raise the bar further—in cryptocurrencies, in the event of credential loss or leak, there is no central authority that can assist in asset recovery. Users thus choose advanced solutions (e.g., [3]), utilizing multiple well-kept cryptographic keys. And even some web services (e.g., the U.S. government website [24]) require multi-factor authentication, while allowing users to combine various credentials like passwords, fingerprints, face recognition, text messages, security keys and more; thus, users choose their own authentication mechanism. At times, the number of credentials is larger than what the user is aware of. For example, online services (e.g., Google [21, 22]) use a device’s location, IP address or cookies as additional credentials. Nevertheless, identity theft incidents are frequent [19], so is loss of Bitcoin [37, 60], as well as digital asset theft, including a recent heist valued at \$600M due to a single company’s key mismanagement [30]. Those illustrate the urgent need to revisit the fundamentals of user authentication.

Previous work (§2) proposed and deployed various authentication solutions. These encompass numerous *credential* types, like passwords, biometrics, and smart cards [6]. But no single credential can be completely relied on [62, 67], leading to the development of advanced mechanisms such as multi-factor authentication (MFA) and interactive protocols [1–3]. However, these works leave open the question of how to model and compare mechanisms’ security. Eyal [18] analyzes a specific type of mechanism, defined by monotonic Boolean functions on presented *credentials* for a small number of credentials. Maram et al. [35] study the authentication problem, but with a bounded-delay communication channel between the mechanism and the user—which is often not available. E.g., when logging in to a web service or issuing small cryptocurrency transactions.

In this work, we study general authentication mechanisms in the common asynchronous setting, with the goal of finding optimally secure mechanisms. First, we define the *asynchronous authentication problem* (§3), where an authentication mechanism interacts with a *user* and an *attacker* (representing all adversarial entities [18, 35]) and tries to identify the user. The mechanism utilizes credentials. Each credential can be known to the user, the attacker, both, or neither. The states of all credentials define a *scenario*. A mechanism is *successful* in a scenario if it correctly identifies the user, for all attacker behaviors and message delivery times. The *profile* of a mechanism is the set of all scenarios in which it is successful.

We seek to reason about a general mechanism’s success when message delivery time is unbounded, yet cryptographic assumptions bound running time according to a security parameter. For simplicity, we first consider ideal credentials that cannot be forged, following similar abstractions in distributed-systems literature [43, 44, 59, 63, 65]. We defer a model that considers cryptographic assumptions to the appendix. Roughly, we require success when the security parameter is large enough, otherwise non-decision is acceptable. This approach may be of independent interest for analyzing distributed protocols that rely on cryptography in asynchronous networks.

To design a mechanism, one must estimate the probabilities of credential faults, e.g., loss or leak [18]. An optimal mechanism has the maximum probability of success given those estimates. To find optimal mechanisms we turn our attention to *Boolean mechanisms*: mechanisms that decide according to a Boolean function of the credentials presented to them. For any authentication mechanism, there exists a Boolean mechanism that dominates it—one that succeeds in the same scenarios or more (§4.1). The proof uses a series of reductions. First, interactive message exchanges do not enhance mechanism security—every interactive mechanism is dominated by

a non-interactive mechanism (assuming as a theoretical construct¹ that credential availability can be proven with a single message). Second, decisions can be made based solely on credential availability proofs. Third, even if randomness were utilized, there exists a dominating deterministic one-shot mechanism. The proofs are constructive and rely on a mechanism’s ability to simulate an execution of another. Finally, for any deterministic mechanism, there exists a dominating mechanism defined by a monotonic Boolean function of the credentials’ availability—a *monotonic Boolean mechanism*.

Let us thus focus on such mechanisms. We show that any monotonic Boolean mechanism defined by a non-constant function is not dominated (maximal), and any maximal mechanism is equivalent (same profile) to a monotonic Boolean mechanism (§4.2). This result shows that studying such mechanisms suffices in the asynchronous setting and reveals a somewhat surprising fact: There is no probability-agnostic hierarchy of Boolean mechanisms in the asynchronous setting unlike under synchrony [35]. For example, with two credentials c_1 and c_2 , consider the mechanism where both c_1 and c_2 are required to authenticate and the mechanism where only c_1 is required. One can easily verify that neither mechanism dominates the other. In fact, this is true in general, for any two non-trivial n -credential Boolean mechanisms.

Since there is no hierarchy of Boolean mechanisms, in general, any such mechanism could be optimal. But evaluating all possible mechanisms, as done by Eyal [18], quickly becomes prohibitively complex, as their number grows super-exponentially with the number of credentials [15].

Instead, we present a scenario-based algorithm to find approximately optimal mechanisms (§5). Given $\delta > 0$, the algorithm finds a δ -optimal mechanism, i.e., a mechanism whose success probability is at most δ lower than that of an optimal mechanism. The algorithm works greedily by building a mechanism that succeeds in scenarios with the highest probability. Once no more scenarios can be added, it removes existing scenarios to add new ones. An analysis of different probability distributions shows that it is sufficient to focus on a small number of high-probability scenarios. Rather than searching all mechanisms, the algorithm stops when the scenarios that might be added have additional probability lower than δ . This is calculated by bounding the number of scenarios in which a mechanism can succeed.

Our algorithm allows for finding approximately-optimal mechanisms with many credentials. This is directly useful for mechanism design. For example, when designing a cryptocurrency wallet, users tend to use a handful of high-quality credentials [36, 40], like hardware keys and long memorized mnemonics. However, we show that adding a few easy-to-lose credentials (e.g., passwords that the user remembers but does not back up) reduces wallet failure probability by orders of magnitude. Similarly, incorporating easy-to-leak credentials (e.g., the name of the user’s pet) with high-quality credentials achieves a similar effect. (Though not in the way such credentials were typically used.)

In summary (§6), our contributions are: (1) formalization of the asynchronous authentication problem; (2) proof that every maximal

mechanism is equivalent to a monotonic Boolean function; (3) proof that any two non-trivial monotonic Boolean mechanisms are either equivalent or neither is better, (4) an approximation algorithm of optimal mechanisms given the credentials fault probabilities, and (5) evaluation of realistic mechanisms and concrete lessons for their improvement.

2 RELATED WORK

Authentication is a fundamental aspect of security [42, 53, 54, 56, 57] that continues to be an active area of research [41, 68]. Whether explicitly or implicitly, previous work mostly addressed two perspectives: credentials and protocols. Credentials are the information used to authenticate a user such as passwords [39], one-time passwords (OTPs) [31], biometrics [66], and physical devices [50]. Protocols define the procedures for authenticating a user based on her credentials. Examples include Kerberos [58] for devices across a network and cryptocurrency wallets [14, 32, 61]. Our focus is on protocols, particularly the abstract mechanisms behind the protocols, which have received only scant attention.

Vashi et al. [62] and Bonneau et al. [6] analyzed various credential types. However, both surveys ultimately concluded that no single credential type offers perfect security, underscoring the importance of protocols that incorporate multiple credentials. Velásquez et al. [64] survey single and multi-factor authentication schemes, examining different credentials combined into multi-factor methods. Zimmermann et al. [67] rate dozens of deployed authentication methods, focusing on user challenges in credential protection rather than the security of the authentication mechanism.

Burrows et al. [11] describe the beliefs of parties involved in authentication protocols as a consequence of communication to realize who has which credentials. Delegation Logic [33] is a logic-based approach to distributed authorization, offering a formal framework for reasoning about delegation relationships. Neither work compares mechanism security, which is the goal of this work.

The importance of authentication mechanisms has grown significantly due to the increasing value of digital assets, such as cryptocurrencies [47] and NFTs [46]. This is especially true in decentralized systems, where users are responsible for securing their assets and, unlike centralized systems, no fallback authority exists. Consequently, numerous cryptocurrency wallets have been developed. The wide variety of authentication mechanisms used in cryptocurrency wallets highlights the need for a formal way to compare them. Nevertheless, previous work focused on credential management and implementation [5, 7].

Hammann et al. [26] uncover vulnerabilities arising from the links between different user credentials and accounts, where one can be used to log into the other. The paper focuses on the connection between different credentials and accounts. However, it does not address the security of different mechanisms, which we do in this work.

Eyal [18] analyzes mechanisms that are Boolean functions of credential availability. We analyze general mechanisms in an asynchronous network and show that all maximally secure mechanisms in this setting can be reduced to mechanisms that are Boolean functions of credentials availability. Eyal finds optimal mechanisms given the probabilities of credential faults for up to 5 credentials using a brute force search and bounds the failure probabilities of

¹In practice interactivity and randomness are useful, e.g., if multiple steps are required to prove credential availability (e.g., challenge-response [48] and interactive proofs of knowledge [20]), and for mechanisms usability and deployability [6] (e.g., a mechanism sending an SMS with a verification code only after verifying a password).

optimal mechanisms using a heuristic approach. Although the brute-force search ensures optionality, it is not computationally feasible for more credentials. We provide an approximation method for near-optimal mechanisms with a large number of credentials based on our observations on the structure of maximally secure mechanisms.

Maram et al. [35] study interactive authentication, where the user and the attacker interact with the mechanism over a synchronous network. They also introduce security profiles of mechanisms, which we use in our analysis. However, unlike our work, the paper assumes a synchronous communication model, which is not practical in many cases, e.g., when processing small payments or logging in to a web service. In such cases, waiting for the true user to receive a message (e.g., sent by email) is unacceptable.

In contrast to the synchronous case, we show that in the asynchronous case, mechanism security does not benefit from interactivity. Our results do not imply that interactive mechanisms are not useful. On the contrary, are widely used in practice, e.g., in multi-factor authentication [1] and challenge-response protocols [48]. Consider for example Google’s security alerts [23] that notify the user of suspicious activity in her account, allowing her to confirm or deny the activity. This is an interactive asynchronous two-factor authentication mechanism, where the user can respond to the mechanism’s request at any time. An equivalent non-interactive mechanism can be one that requires both the user’s password and her fingerprint to authenticate. However, in this case, the non-interactive mechanism is less convenient for the user. For every interactive mechanism, we show a theoretical reduction to a non-interactive mechanism that succeeds in the same scenarios or more. All maximally secure mechanisms have equivalent mechanisms that are Boolean functions of credentials availability, which is not the case in the synchronous model.

Although our analysis uses methods similar to those used in distributed systems’ theory like asynchronous communication and simulations, the authentication problem is distinct from classical problems like consensus or broadcast [16]: Success is defined by the decision of a single party and not multiple ones, and credentials take a central role.

3 MODEL

We formalize the asynchronous authentication problem (§3.1), specifying an execution, its participants, and their behavior. Then we define mechanism success and the relation between different mechanisms (§3.2).

3.1 Asynchronous Authentication

The system comprises three entities, an *authentication mechanism* M and two *players*: a *user* U and an *attacker* A . All three are finite deterministic automata that can draw randomness from a *random tape* v , an infinite stream of random bits. We specify the automata as computer programs [52].

An execution is orchestrated by a *scheduler*, whose pseudocode is given in Appendix A.2. Each of the players interacts with the mechanism by sending and receiving messages. The scheduler, parametrized by $\gamma_{ID} \in \{0, 1\}$, assigns the user identifier γ_{ID} and the attacker identifier $(1 - \gamma_{ID})$. E.g., if $\gamma_{ID} = 0$, then the user is player 0 and the attacker is player 1. The identifiers serve a similar purpose to that of cookies that identify a website visitor during a single

session. They allow the mechanism to identify and distinguish between the players during an execution without revealing which is the user.

To facilitate authentication, the players take advantage of private information to convince the mechanism that they are the user. This information is a set of credentials that might be data (e.g., passwords), biometric properties (e.g., fingerprints), and physical objects (e.g., phones or smart cards). Each credential has two parts: a public part, known to the mechanism, and a secret part. For example, when using a password to authenticate to a website, the user remembers the password (the secret part) and the website stores its hash (the public part); or when using public key authentication, a blockchain has the public key (the public part) and the user has the private key (the secret part).

DEFINITION 1 (CREDENTIAL). A credential c is a tuple $c = (c^P, c^S)$ where c^P is the public part of the credential and c^S is the secret part of the credential.

The system uses a set of n credentials, $\{c_1, \dots, c_n\}$. A credential can be *available* to a player. An *availability vector* represents the availability of all credentials to a player. Each entry i in the vector indicates the availability of credential i . Denote by σ_U, σ_A the availability vector of the user and the attacker respectively. A scenario σ is a tuple of availability vectors (σ_U, σ_A) . Denote by C_U^σ, C_A^σ the set of credentials available to the user and attacker in scenario σ respectively. The attacker is polynomially bounded [13, 28] and cannot guess the secret parts of the credentials better than a random guess in a polynomial number of steps, except with negligible probability.

The system uses a credential generation function $gen(n)$ that generates a set of n credentials $\{(c_i^P, c_i^S)\}_{i=1}^n$. Anyone with the secret part of a credential can prove its availability to the mechanism, and anyone without the secret part cannot forge a proof of its availability. In practice, cryptographic assumptions require bounding the execution time based on a security parameter [49]. However, in the asynchronous system, message delivery time is unbounded, and so is execution time—the mechanism is only required to eventually decide.

For simplicity, we first assume that credentials are *ideal*, that is, that their cryptographic guarantees are maintained indefinitely. This is a common assumption in distributed-systems literature, e.g., [43, 44, 59, 63, 65]. We properly reconcile the tension between the models in Appendix A; intuitively, the mechanism is only required to succeed when the security parameter is large enough. The results hold for this model as well, as shown in Appendix B.

An *authentication mechanism* exchanges messages with the players to decide which is the user.

DEFINITION 2 (AUTHENTICATION MECHANISM). An authentication mechanism M is a finite deterministic automaton specified by two functions, $gen_M(\cdot)$ and $step_M(\cdot)$:

- $gen_M(n)$, where n is the number of credentials, is an ideal credential generator. And
- $step_M(msg, i)$, where msg is the message the mechanism received (perhaps no message, represented by $msg = \perp$), sent from player i (if no message was received, $i = \perp$), is a function that updates the state of the mechanism and returns a pair (msg_0, msg_1) of sets of messages (maybe empty) to

send to the players by their identifiers. It can access and update the mechanism's state and the set of credentials public part $\{c_i^p\}_{i=1}^n$.

The mechanism uses a variable $decide_M$ that marks which identifier it decides belongs to the user. The variable's initial value is \perp . When M reaches a decision, it updates the variable to an identifier value.

A player is defined by its strategy, a function that defines its behavior. Formally,

DEFINITION 3 (STRATEGY). A strategy S of a player $p \in \{U, A\}$ is a function that takes a message from the mechanism as input (maybe empty, denoted by \perp), and has access to the player's state and credentials C_p^σ . It updates the player's state and returns a set of messages to be sent back to the mechanism.

We now describe the execution of the system. Full details including pseudocode are in Appendix A.2. The execution consists of two parts: *setup* (Setup function) and a *main loop* (Execute function). The setup generates a set of n credentials using the function $gen_M(n)$, and assigns each player credentials according to the scenario σ . Then it assigns an identifier to each player based on $\gamma_{ID} \in \{0, 1\}$. Whenever a random coin is used by M or the players, the result is the next bit of v .

Once the setup is complete, the main loop begins and both players $p \in \{U, A\}$ can send messages to the mechanism, each based on her strategy S_p and set of credentials C_p^σ . Time progresses in discrete steps. The communication network is reliable but asynchronous. Messages arrive eventually, but there is no bound on the time after they were sent and no constraints on the arrival order.

This is implemented as follows. The scheduler maintains three sets of pending messages, one for each entity. The scheduler is parametrized by an ordering function γ_{ord} and a random tape v_γ . In each step, it uses γ_{ord} to choose a subset of messages (maybe empty) from each of the pending message sets, and returns them as an ordered list. The scheduler's random coins used in γ_{ord} are taken from v_γ . The scheduler removes the chosen messages from the pending message sets.

Each player receives the messages chosen by the ordering function γ_{ord} and sends messages to the mechanism according to her strategy. These messages are added to the mechanism's pending messages set. Similarly, the mechanism receives the messages chosen by γ_{ord} . After every message msg it receives from the player with identifier $i \in \{0, 1\}$, the mechanism runs its function $step_M(msg, i)$ and sends messages to the players.

In each step, the scheduler checks if $decide_M \neq \perp$, meaning that the mechanism has reached a decision. Once it does, the execution ends and the player with the matching index (either 0 or 1) wins. The tuple $(\gamma_{ID}, \gamma_{ord}, v_\gamma)$ thus defines the scheduler's behavior. And an execution E is thus defined by its parameter tuple $(M, \sigma, S_U, S_A, \gamma, v)$; by slight abuse of notation we write $E = (M, \sigma, S_U, S_A, \gamma, v)$.

We define the *winner* of an execution as the player with the identifier that the mechanism decides in that execution.

3.2 Mechanism Success

A mechanism is *successful* in a scenario σ if the user wins against all attacker strategies and schedulers. Formally,

DEFINITION 4 (MECHANISM SUCCESS). A mechanism M is successful in a scenario σ if there exists a user strategy S_U such that for

all attacker strategies S_A , schedulers γ , and random tapes v , the user wins the execution $E = (M, \sigma, S_U, S_A, \gamma, v)$. Such a user strategy S_U is a winning user strategy in σ with M . Otherwise, the mechanism fails.

The set of scenarios in which the mechanism succeeds is its *profile* (as Maram et al. [35] defined for a synchronous network).

DEFINITION 5 (PROFILE [35]). A profile is a set of scenarios. The profile of a mechanism M , denoted $\Pi(M)$, is the set of all scenarios in which M succeeds.

We are now ready to define the *asynchronous authentication problem*.

DEFINITION 6 (ASYNCHRONOUS AUTHENTICATION). Given a profile π , a mechanism M solves the π asynchronous authentication problem if M is successful in all scenarios in π (maybe more), i.e., $\pi \subseteq \Pi(M)$.

In the asynchronous setting, the profile defines a relation between any two mechanisms M_1 and M_2 with the same number of credentials. The following definition is similar to that given by Maram et al. [35] for a synchronous network, but as we will consider individual credential probabilities, our definition is permutation sensitive.

DEFINITION 7 (MECHANISMS ORDER). A mechanism M_1 dominates (resp., strictly dominates) a mechanism M_2 with the same number of credentials if the profile of M_1 is a superset (resp., strict superset) of the profile of M_2 : $\Pi(M_2) \subseteq \Pi(M_1)$ (resp., $\Pi(M_2) \subset \Pi(M_1)$).

Two mechanisms M_1 and M_2 are equivalent if they have the same profile $\Pi(M_1) = \Pi(M_2)$. And incomparable if neither dominates the other $\Pi(M_1) \not\subseteq \Pi(M_2)$ and $\Pi(M_2) \not\subseteq \Pi(M_1)$.

4 MAXIMAL MECHANISMS

Having defined partial ordering on mechanisms, we proceed to identify *maximal mechanisms*, i.e., mechanisms that are not strictly dominated. We show that for any mechanism there exists a dominating mechanism that is a Boolean function of the credentials' availability (§4.1). Then we show that mechanisms defined by monotonic Boolean functions are maximal, and all maximal mechanisms are equivalent to a Boolean mechanism (§4.2).

4.1 Domination by Boolean Mechanisms

We show that any mechanism is dominated by a mechanism that is a Boolean function of the credentials' availability. We take 4 similar steps. First, we show that for all mechanisms there exists a dominating *one-shot* mechanism that decides based on up to a single message from each player (§4.1.2). Second, we show that decisions can be made solely on credential availability proofs (§4.1.3). Then we show that randomness does not improve the security of a one-shot mechanism (§4.1.4). Finally, we show that for any one-shot, deterministic mechanism, there exists a dominating mechanism defined by a monotonic Boolean function of the credentials' availability (§4.1.5).

NOTE 1 (PRACTICALITY OF ONE-SHOT AND DETERMINISTIC MECHANISMS). Numerous interactive and randomized mechanisms are practical and widely used, e.g., challenge-response protocols [48] and interactive proofs of knowledge [20]. While we prove a reduction

from any mechanism to a one-shot and deterministic mechanism, the propositions below are for theoretical purposes and do not necessarily imply that such mechanisms are practical nor suggest that they should replace interactive, randomized mechanisms.

We rather show that, security-wise, for every interactive or random mechanism, there exists a non-interactive deterministic mechanism that succeeds in the same scenarios or more. This is only used as a proof step of the reduction to monotonic Boolean functions. However, the non-interactive mechanism might be less practical. E.g., a 2FA mechanism that first requires a password and only when the password is correct, it asks for a one-time password sent to the user's phone, is more practical than a mechanism that requires the user to send both the password and the one-time password in a single message, as it saves the company the cost of sending the one-time password in case the password is incorrect.

4.1.1 Step template. Each step in our proof has the following parts: Given a mechanism M_1 we show it is dominated by a mechanism M_2 with a certain property. We prove constructively by defining a mechanism M_2 and show the required property holds. The domination proof takes advantage of a mechanism's ability to simulate the execution of another mechanism. That is, the mechanism M_2 takes a mechanism M_1 , two strategies S_0 and S_1 , the sets of credentials each strategy uses C_0 and C_1 , an ordering function γ_{ord} , a scheduler random tape v_γ , a random tape v , and a number of steps t , and runs $Execute(M_1, S_0, C_0, S_1, C_1, \gamma_{ord}, v_\gamma, v, t)$ (Appendix A.2). Note that when calling the function $Execute$, if t is not explicitly given, it is assumed to be unbounded. The simulation result is the identifier that $decide_{M_1}$ gets during the execution of $Execute$, if it terminates with a decision, and \perp otherwise.

NOTE 2 (MECHANISM'S COMPUTATIONAL RESOURCES). In the following proofs, we assume the constructed mechanism has sufficient computational resources to simulate the original mechanism in a single step. At the end of this section, we show that there always exists a dominating polynomial mechanism.

NOTE 3 (PLAINTEXT CREDENTIALS). In the following proofs, we assume the user sends the secret part of her credentials (as with passwords). Although this is not always the case (e.g., with cryptographic signatures), this is only a theoretical construction and does not suggest that the user should generally send her secrets in plaintext. In practice, she must only be able to prove she can access them.

NOTE 4 (MECHANISM AS A FUNCTION). When constructing a mechanism M_{new} given a mechanism M , we use the notation $M_{new}(M)$ to denote the mechanism M_{new} is a function of M . When it is clear from the context, we omit the argument M and write M_{new} .

4.1.2 One-shot Mechanisms. We define a one-shot (OS) mechanism as one that, in every execution, reaches a decision based only on the first message it receives (if any) from each player. If a player sends multiple messages, the mechanism ignores all but the first it receives.

DEFINITION 8 (ONE-SHOT MECHANISM). A mechanism M is a one-shot mechanism if for all scenarios σ , schedulers γ , and random tapes v , and for all user strategies S_U, S'_U and attacker strategies S_A, S'_A such that the first message that M receives from each player (if any) in the executions $E_1 = (M, \sigma, S_U, S_A, \gamma, v)$ and $E_2 = (M, \sigma, S'_U, S'_A, \gamma, v)$

Algorithm 1: $M_{OS}(M)$'s step function

```

step $_{M_{OS}}$ (msg, i)
1 for  $j \in \{0, 1\}$  do
2   if processing $_j = 1$  then // A message already received from j
3     ( $S_j, C_j, v_{\gamma, j}, v'_j$ )  $\leftarrow$  Sim $_j$  // Read simulation params
4      $S_{1-j}, C_{1-j} \leftarrow \perp, \perp$ 
5     decide $_{M_{OS}} \leftarrow Execute(M, S_0, C_0, S_1, C_1, \gamma_{ord}, v_{\gamma, j}, v'_j, t)$  // Simulate  $M$ 's execution
6   else if  $j = i$  then // First message received from i
7     processing $_i \leftarrow 1$  // Mark receiving a message from j
8      $S_i, C_i \leftarrow extractStrategy(msg)$  // Extract strategy
9      $S_{1-i}, C_{1-i} \leftarrow \perp, \perp$ 
10    if  $S_i = \perp$  then // Invalid strategy
11      decide $_{M_{OS}} \leftarrow 1 - i$  // The other player wins
12    else
13      Sim $_i \leftarrow (S_i, C_i, v_{\gamma}, v')$  // Save simulation params
14      decide $_{M_{OS}} \leftarrow Execute(M, S_0, C_0, S_1, C_1, \gamma_{ord}, v_{\gamma}, v', t)$  // Simulate  $M$ 's execution
15  return  $\perp, \perp$  // No messages to send

```

is the same and in the same order, then M reaches the same decision or does not decide in both executions.

Given any mechanism, we construct a dominating one-shot mechanism by simulating the original mechanism's execution.

CONSTRUCTION 1. We define M_{OS} by specifying the functions $gen_{M_{OS}}(\cdot)$ and $step_{M_{OS}}(\cdot)$. The credentials' generation function $gen_{M_{OS}}(\cdot)$ is the same as $gen_M(\cdot)$.

The mechanism's step function proceeds as follows (Algorithm 1.) If it does not receive a message during its execution, it does nothing. If it receives multiple messages from a player, it ignores all but the first one (lines 2 and 7). If M_{OS} receives a message that is not a valid strategy and credentials pair, then it decides the identifier of the other player (line 11).

Consider the first message it receives, and let $t_1 \in \mathcal{N}^+$ be the step in which it arrives. If the message is an encoding of a valid strategy and credentials pair, then M_{OS} simulates an execution of M (line 14) with the given strategy and credentials while setting both the opponent's strategy and credentials to \perp each (lines 8-9). It uses a scheduler random tape v_γ and an execution random tape v' drawn from v , and an ordering function $\gamma_{ord}^{v_\gamma}$ that chooses the time and order of message delivery randomly based on v_γ . The simulation runs for t_1 steps. If M 's simulated execution decides then M_{OS} decides the same value. Otherwise, M_{OS} saves the above execution details (line 13) and waits for the next message.

In each time step $t' > t_1$ between the message arrival from the first player and the message arrival from the second player (might be infinite), M_{OS} reads the simulation parameters it saved (line 3) and runs the same simulation with the exact same parameters as before but for $t' \in \{t_1 + 1, t_1 + 2, \dots\}$ steps each time (line 5). If the simulation reaches a decision, then M_{OS} decides the same value.

If a message arrives from the other player at some time $t_2 > t_1$, then, similar to the previous case, M_{OS} simulates an execution of M with the given strategy and credentials while setting the opponent's to \perp for t_2 steps. And again, if M 's simulated execution decides, then M_{OS} decides the same value.

Otherwise, M_{OS} saves the above execution details (line 13) and continues to simulate the execution of M for $t'' \in \{t_2 + 1, t_2 + 2, \dots\}$ steps. In each of its steps, M_{OS} runs two simulations of M 's execution, one with the first player's simulation parameters and the other with the second player's simulation parameters, both for t'' steps (line 5). Once a simulation decides, then M_{OS} decides the same value.

The mechanism $M_{OS}(M)$ is one-shot as it decides based only on the first message it receives from each player. To prove it dominates M , we first show that the attacker's message does not lead to her winning.

LEMMA 1. *Let σ be a scenario and let M be a mechanism successful in σ . Then, for all executions of $M_{OS}(M)$ in scenario σ in which the function $step_{M_{OS}(M)}(\cdot)$ receives a message from the attacker for the first time, either the function sets $decide_{M_{OS}}$ to the user's identifier or the simulated execution of M does not decide.*

PROOF. Let σ be a scenario and let M be a mechanism successful in σ . Consider an execution of M_{OS} in σ in which the function $step_{M_{OS}}(\cdot)$ receives an attacker's message for the first time. Let t be the time step in which the message arrives and denote the identifier of the attacker by $i \in \{0, 1\}$. If the attacker's message is not a valid encoding of a strategy and credentials set, then M_{OS} decides the identifier of the user, and we are done.

Otherwise, the attacker's message is an encoding of a valid strategy and credentials pair (S_A, C_A) . In this case, M_{OS} sets the strategies and credentials to $S_i = S_A$, $C_i = C_A$, $S_{1-i} = \perp$, and $C_{1-i} = \perp$, $\gamma_{ord}^{v_Y}$, v_Y and v' as in the definition of M_{OS} , and simulates M 's execution by running $Execute(M, S_0, C_0, S_1, C_1, \gamma_{ord}^{v_Y}, v_Y, v', t)$. If the simulation returns the user's identifier or does not decide for any t , we are done. The only remaining option is that the simulation returns the identifier of the attacker. To show this is impossible, assume by contradiction that there exists a $t' \geq t$ for which the attacker wins in this simulated execution.

First, we show that there exists an execution of M that is identical to the simulated one. Let $\gamma = (\gamma_{ID}^{OS}, \gamma_{ord}^{v_Y}, v_Y)$ be a scheduler such that the user gets the same identifier as in the execution of M_{OS} , with the same ordering function and scheduler random tape that M_{OS} uses to simulate M 's main loop. And let v_M be a random tape such that when the execution $E = (M, \sigma, \perp, S_A, \gamma, v_M)$ reaches the main loop, all the next bits of v_M are equal to v' . Note that the main loop of E is the same as the one M_{OS} simulates in E_{OS} . And the attacker wins in the execution E (by the contradiction assumption).

We thus established a simulated execution in which the attacker wins, and in this simulation, there exists a time $\tau \leq t'$ when the simulated M decides the identifier of the attacker. Since M is successful in scenario σ , there exists a winning user strategy S_U . Let γ' be a scheduler such that in the execution $E' = (M, \sigma, S_U, S_A, \gamma', v_M)$ it behaves like γ except M receives the user's messages not before τ . Such a scheduler exists since the communication is asynchronous and message delivery time is unbounded.

At any time step before τ , the mechanism M sees the same execution prefix whether it is in the execution E with an empty user strategy or in the execution E' with a winning user strategy. Thus, it cannot distinguish between the case where it is in E or E' at τ . Since in E the mechanism M decides at τ , M must decide the same value at τ in E' . That is, the attacker wins also in the execution E' , contradicting the fact that S_U is a winning user strategy for M in σ . Thus, the attacker cannot win in the simulated execution of M . \square

Now we can prove domination.

PROPOSITION 1. *For all profiles π , an authentication mechanism that solves the π asynchronous authentication problem is dominated by a one-shot mechanism.*

PROOF. Assume that M is successful in a scenario σ . Then, there exists a user strategy S_U such that for every attacker strategy S_A , scheduler γ , and random tape \tilde{v} , the user wins the corresponding execution $(M, \sigma, S_U, S_A, \gamma, \tilde{v})$.

We now show that M_{OS} is successful in scenario σ as well. Denote by S_U^{OS} the user strategy that sends an encoding of the winning user strategy S_U and a set of credentials $C_U \subseteq C_U^\sigma$ that S_U uses in a single message on the first step. And consider any execution of M_{OS} in scenario σ with the user strategy S_U^{OS} . Note: In the *Setup* function of an execution, the first player in the tuple is always the user and the second is the attacker. However, in the *Execute* function, the first player is the player with identifier 0, which can be the user or the attacker, and the second is the player with identifier 1.

As S_U^{OS} sends a message in the first step, the user's message eventually arrives at some time step t . The mechanism receives and processes at least one message during its execution; we consider two cases separately: a user's message arrives first, or an attacker's message arrives first.

Denote by $i \in \{0, 1\}$ the identifier of the player whose message arrives first and by t_1 its arrival time. If the user's message msg arrives first with the encoded strategy and credential set $(S_U, C_U) = extractStrategy(msg)$, then M_{OS} sets the strategies to $S_i = S_U$, $C_i = C_U$, $S_{1-i} = \perp$, and $C_{1-i} = \perp$, the scheduler random tape γ_v , the ordering function $\gamma_{ord}^{v_Y}$ and the random tape v' as described in M_{OS} 's definition, and simulates M 's execution by running $Execute(M, S_0, C_0, S_1, C_1, \gamma_{ord}^{v_Y}, v_Y, v', t_1)$. Denote by $\gamma = (\gamma_{ID}^{OS}, \gamma_{ord}^{v_Y}, v_Y)$ the scheduler such that the user gets the same identifier as in the execution of M_{OS} , with the same ordering function and scheduler random tape that M_{OS} uses to simulate M 's main loop. Because S_U is a winning user strategy in M , the user wins the execution $E = (M, \sigma, S_U, \perp, \gamma, v')$. Therefore, there exists a time step τ such that M decides the user at τ . If $t_1 \geq \tau$, then M_{OS} decides the user as well. Otherwise, $t_1 < \tau$, then M_{OS} repeatedly simulates M 's execution for $t' \in \{t_1 + 1, t_1 + 2, \dots\}$ steps. If no other message arrives before τ , then once it reaches $t' = \tau$, M_{OS} decides the user as well. Otherwise, if the attacker's message arrives at some $t_2 < \tau$, and by Lemma 1, either M_{OS} decides the user or the simulated execution of M with the attacker's strategy and credentials does not decide. Again, if M_{OS} decides the user, then we are done. Otherwise, M_{OS} continues to simulate both executions of M with the user's and attacker's simulations parameters, respectively, for $t'' \in \{t_2 + 1, t_2 + 2, \dots\}$ steps. Finally, at $t'' = \tau$, the simulated execution of the user's strategy decides the user, so M_{OS} decides the user as well.

Now assume the attacker's message arrives first to M_{OS} at time t_1 . Again by Lemma 1, either M_{OS} decides the user or the simulated execution of M in σ with the attacker's strategy does not decide. If M_{OS} decides the user, then we are done. Otherwise, the simulation does not decide for all $t \geq t_1$, and M_{OS} waits for the next message from the other player (the user). As long as no other message arrives, M_{OS} keeps simulating the same execution in each step for a longer time but does not reach a decision.

Because the user's message must eventually arrive, M_{OS} receives the user's message with her encoded strategy and credentials at time $t_2 > t_1$. Then similar to the previous case, M_{OS} sets the parameters as described in Construction 1, and simulates M 's execution. By the same argument as before, we get that once $t_2 \geq \tau$, the simulation terminates and returns the user's identifier, so M_{OS} also returns it.

Overall we showed that the user wins in all executions of M_{OS} in scenario σ . Thus, it is successful in σ . And we conclude that the one-shot mechanism M_{OS} dominates M . \square

We illustrate Proposition 1 with an example.

EXAMPLE 1. Consider a mechanism M^{ex} with two credentials: a password c_1 and an OTP c_2 . Authentication with M^{ex} has three steps: First, the player clicks "start". Then, she enters her username and password. If those are correct, she is asked to enter the OTP. If all three steps are done successfully, the player wins and gets authenticated. In case no player wins, the mechanism chooses a player at random.

For any player to authenticate, she must complete all three steps successfully. This not only requires knowing the correct credentials but also requires that all messages arrive before the mechanism decides. However, even if the user sends the correct credentials, some of the messages might be delayed, and as the mechanism must decide, it might do so before receiving all messages. In this case, the mechanism chooses a player at random, and the user might not authenticate. Therefore, this mechanism's profile is empty $\Pi(M^{ex}) = \emptyset$.

A dominating one-shot mechanism M_{OS}^{ex} has the player send the content of all three steps in a single message. The rest of the decision process of the mechanism stays the same. If the player sends the content of all three steps in a single message, then she wins. Therefore, if the user knows both credentials, she can authenticate. However, if the user knows only one of the credentials, she cannot always authenticate. And if the attacker knows at least one of the credentials, she can authenticate in some cases (depending on the mechanism's random choice). Therefore, the mechanism's profile is the scenario in which the user knows both credentials and the attacker knows none $\Pi(M_{OS}^{ex}) = \{((1, 1), (0, 0))\} \supset \Pi(M^{ex})$.

4.1.3 Credential-based mechanisms. A one-shot mechanism is a credential-based mechanism if it reaches a decision based only on a function of the credentials it receives from players.

DEFINITION 9 (CREDENTIAL-BASED MECHANISM). A mechanism M is a credential-based mechanism if it is a one-shot mechanism and for all scenarios σ , schedulers γ and random tapes v , and for all user and attacker strategies S_U, S'_U, S_A and S'_A such that S_U and S_A send only messages that are subsets of the corresponding player's credentials' vector, and S'_U and S'_A send messages with the same subsets of credentials but with any additional strings, then M reaches the same decision in both executions $E_1 = (M, \sigma, S_U, S_A, \gamma, v)$ and $E_2 = (M, \sigma, S'_U, S'_A, \gamma, v)$.

LEMMA 2. For all one-shot mechanisms M_{OS} there exists a credential-based mechanism M_{cred} that dominates M_{OS} .

The proof of Lemma 2 is similar to that of Proposition 1 and is omitted for brevity. The key difference is the observation that any

message a player can send is a function of (1) common information [25] available to everyone, (2) credentials of the sender, and (3) random bits.

Denote by M_{OS} a one-shot mechanism. If a player has a winning strategy for M_{OS} , knowing some common information, her credentials, and the random bits, she can generate the messages she needs to send to win in the credential-based mechanism as well.

Formally, we define a function $S_{OS}(\cdot)$ that maps a set of credentials to a strategy. Let c be a set of credentials known to player $i \in \{0, 1\}$, the function $S_{OS}(c)$ returns a strategy for player i for M_{OS} using c such that if the first message that M_{OS} receives is from player i , it leads to player i winning the execution. If no such a strategy exists, $S_{OS}(c) = \perp$. Then the construction of a credential-based mechanism M_{cred} is similar to that of M_{OS} with the following change: Instead of extracting the strategy and credentials from the message, M_{cred} uses the function $S_{OS}(\cdot)$ to obtain the strategy given the credentials.

The mechanism M_{cred} is credential-based as is one-shot and decides based only on the credentials it receives.

EXAMPLE 2. Following Example 1, a dominating credential-based mechanism M_{cred}^{ex} of M_{OS}^{ex} has the player send only her password and OTP in a single message. The rest of the decision process of the mechanism remains the same. The pressing of "start", which is not a credential, is redundant.

In practice, the username is necessary to authenticate; however, it is not a credential (not a secret.) For simplicity, we abstract it away and assume the mechanism knows in advance which account the authentication is for.

4.1.4 Deterministic mechanisms. Next, we show that in one-shot mechanisms, using randomness to determine if a player is the user or not, does not help the mechanism design.

LEMMA 3. For all one-shot credential-based mechanisms M_{cred} there exists a deterministic credential-based mechanism M_{det} that dominates M_{cred} .

Denote by M_{cred} a credential-based authentication mechanism. A deterministic mechanism M_{det} that behaves the same as M_{cred} except that its step function does not use any randomness can be constructed by artificially replacing the randomness used in $step_{M_{cred}}(\cdot)$ with a stream of zeroes. Thus, preventing M_{det} from using any randomness in its step function and making it deterministic. Again, the proof of Lemma 3 is similar to that of Proposition 1 and Lemma 2 and is omitted for brevity.

EXAMPLE 3. Following Example 2, a dominating deterministic credential-based mechanism M_{det}^{ex} requires both the password and the OTP in a single message. If a player sends the correct password and OTP, then she wins. Otherwise, the other player wins (instead of randomly choosing a winner). The profile of M_{det}^{ex} is $\Pi(M_{det}^{ex}) = \{((1, 1), (0, 0)), ((1, 1), (1, 0)), ((1, 1), (0, 1))\} \supset \Pi(M_{cred}^{ex})$. That is, all scenarios where both credentials are available to the user and not to the attacker.

4.1.5 Boolean mechanisms. We show that every deterministic credential-based mechanism is dominated by a mechanism defined by a monotonic Boolean function.

DEFINITION 10 (BOOLEAN MECHANISM). *Given a Boolean function on n variables $f : \{0, 1\}^n \rightarrow \{0, 1\}$, we define the Boolean mechanism M_f as follows: The credential generation function $gen_{M_f}(\cdot)$ is a secure credential generation function. The step function $step_{M_f}(msg, i)$ receives a single message msg from a player with identifier i that contains a set of credentials c . The mechanism M_f extracts the credentials' availability vector q from c . If c is not a valid set of credentials, M_f decides the identifier of the other player $1 - i$. If $f(q) = 1$, M_f decides i . Otherwise, $f(q) = 0$ and the mechanism decides $1 - i$. The mechanism M_f is the Boolean mechanism of f . If f is monotonic, then M_f is the monotonic Boolean mechanism of f .*

Intuitively, for a deterministic credential-based mechanism, the only information that matters is whether a player can prove she has a set of credentials. If a player can authenticate to a mechanism, accessing additional credentials will not prohibit her from authenticating to the same mechanism.

CONSTRUCTION 2. *Denote by M_{det} a deterministic credential-based mechanism with n credentials. Define $f : \{0, 1\}^n \rightarrow \{0, 1\}$ as follows: For all $q \in \{0, 1\}^n$:*

$$f(q) = \begin{cases} 1 & q \in \{\sigma_U \in \{0, 1\}^n \mid \exists \sigma = (\sigma_U, \sigma_A) \in \Pi(M_{det})\} \\ 0 & \text{otherwise} \end{cases}$$

f is the function derived from M_{det} . M_f is the mechanism defined by f , with the function $gen_{M_f}(\cdot) := gen_{M_{det}}(\cdot)$. It is easy to see that f is well-defined. It remains to show that it is monotonic and that it dominates M_{det} .

LEMMA 4. *Let M_{det} be a deterministic credential-based mechanism and let the function f be as constructed above. Then f is monotonic.*

PROOF. Given two binary vectors $x, y \in \{0, 1\}^n$, we denote $x \geq y$ if for every index $i \in [n]$, $x_i \geq y_i$. In other words, if an element is set to 1 in y , it is also set to 1 in x .

We now prove that the function f is monotonic. Let $x, y \in \{0, 1\}^n$ such that $y \geq x$ and $f(x) = 1$. By definition of f , $f(x) = 1$ if and only if there exists a scenario σ in which the user's availability vector is x and M_{det} succeeds in σ . Let S_U be the winning user strategy for M_{det} in σ .

As M_{det} is a credential-based mechanism, S_U sends only a set of credentials c_U in a single message. Now consider the scenario σ' in which the user's availability vector is y and the attacker's availability vector is the same as in σ . The user can use the exact same strategy S_U in σ' , as she has access to all credentials she has in σ and more. Thus, for all attacker strategies S_A , schedulers γ , and random tapes v , the user wins the execution $E = (M_{det}, \sigma', S_U, S_A, \gamma, v)$. Therefore, M_{det} succeeds in σ' as well, $f(y) = 1$ and f is monotonic. \square

To prove Lemma 4, we show that if M_{det} is successful in a scenario σ , then M_f is successful in σ as well. We build on the observation that based on the scheduler, an execution of M_f has 3 possible paths: either (1) M_f receives no messages, thus it does not decide. Or (2) the user's message arrives first, with her set of credentials corresponding to the availability vector σ_U . As M_{det} is successful in σ , by definition of f we get that $f(\sigma_U) = 1$ and M_f decides the user.

The last possibility is that (3) the attacker's message arrives first, if it is not a valid set of credentials, then M_f decides the user. Otherwise, if it is a valid set of credentials, we show that it is not possible that f evaluates to 1 on the attacker's availability vector. The proof is somewhat similar to that of Lemma 1 and is done by contradiction. If f evaluates to 1 on the attacker's availability vector, then there exists two execution prefixes of M_{det} that are indistinguishable to the mechanism M_{det} and lead to the attacker winning. In one of which the attacker wins, although the user uses her winning strategy, contradicting the fact that M_{det} is successful in σ . Thus, M_f decides the user. As in all cases, if a message arrives to M_f , then M_f decides the user. We conclude that M_f is a monotonic Boolean mechanism that dominates M_{det} . The proof can be found in Appendix C.

EXAMPLE 4. *Consider the deterministic credential-based mechanism M_{det}^{ex} from Example 3. A dominating monotonic Boolean mechanism M_f^{ex} is the mechanism defined by the function $f(c_1, c_2) = c_1 \wedge c_2$.*

4.1.6 Any mechanism is dominated by a Boolean one. We now show that every mechanism is dominated by a monotonic Boolean mechanism.

THEOREM 1. *For all profiles π , for all mechanisms M that solve the π -asynchronous authentication problem, there exists a monotonic Boolean mechanism M_f that dominates M .*

PROOF. Let π be a profile and let M be a mechanism that solves the π asynchronous authentication problem. By Proposition 1 there exists a one-shot mechanism M_{OS} dominating M , from Lemma 2 there exists a credential-based mechanism M_{cred} dominating M_{OS} , and by Lemma 3, M_{cred} is dominated by a deterministic credential-based mechanism M_{det} . Finally, by Lemma 4 there exists a monotonic Boolean mechanism that dominates M_{det} . Overall we get that any mechanism is dominated by a monotonic Boolean mechanism. \square

Note that it is not true that for every mechanism there exists an equivalent Boolean mechanism. We show an example of a mechanism that has no equivalent Boolean mechanism.

EXAMPLE 5. *Consider the mechanism M_{OS}^{ex} from Example 1. We showed that, $\Pi(M_{OS}^{ex}) = \{((1, 1), (0, 0))\}$. However, there exists no Boolean mechanism with the same profile. There are six monotonic Boolean functions with two variables. Two of which are the constant functions $f_1(v) = 0$ and $f_2(v) = 1$ for all $v \in \{0, 1\}^2$ and one can easily confirm that their profiles are empty. The other four functions are $f_3(c_1, c_2) = c_1$, $f_4(c_1, c_2) = c_2$, $f_5(c_1, c_2) = c_1 \vee c_2$, and $f_6(c_1, c_2) = c_1 \wedge c_2$. None of these functions has a profile that includes only a single scenario.*

4.2 Monotonic Boolean Mechanisms are Maximal

We now prove that every non-trivial monotonic Boolean mechanism is maximal. First, we show that a Boolean mechanism is successful in a scenario (σ_U, σ_A) if and only if the user has sufficient credentials and the attacker does not, that is, $f(\sigma_U) = 1$ and $f(\sigma_A) = 0$. The proof can be found in Appendix D.

OBSERVATION 1. Let M_f be a monotonic Boolean mechanism of the function f . Let $T = \{v \in \{0, 1\}^n \mid f(v) = 1\}$ and let $F = \{v \in \{0, 1\}^n \mid f(v) = 0\}$. The profile of M_f is the Cartesian product of T and F .

$$\Pi(M_f) = \{(\sigma_U, \sigma_A) \in \{0, 1\}^n \times \{0, 1\}^n \mid \sigma_U \in T, \sigma_A \in F\}.$$

And the profile's size is $|\Pi(M_f)| = |T| \cdot |F|$.

We intend to gradually build the mechanism's function. To this end, we recall the definition of a partially-defined Boolean function—a function that defines output values for a subset of the Boolean vectors.

DEFINITION 11 (PARTIALLY-DEFINED BOOLEAN FUNCTION). [9] A partially-defined Boolean function (or partial Boolean function) is a pair of disjoint sets (T, F) of binary n -vectors. The set T denotes the set of true vectors and F denotes the set of false vectors. A Boolean function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ is called an extension of (T, F) if $f(x) = 1$ for all $x \in T$ and $f(y) = 0$ for all $y \in F$.

We extend the definition of a Boolean mechanism to a partial Boolean mechanism.

DEFINITION 12 (PARTIAL BOOLEAN MECHANISM). Given a partial Boolean function on n variables (T, F) , we define the mechanism $M_{(T,F)}$ as follows: The credential generation function $\text{gen}_{M_{(T,F)}}(\cdot)$ is a secure credential generation function. The step function $\text{step}_{M_{(T,F)}}(\text{msg}, i)$ receives a single message msg from a player with identifier i that contains a set of credentials c . The mechanism $M_{(T,F)}$ extracts the credentials' availability vector q from c . If c is not a valid set of credentials, $M_{(T,F)}$ decides the identifier of the other player $1 - i$. If $q \in T$, the mechanism $M_{(T,F)}$ decides i . And if $q \in F$, the mechanism decides $1 - i$. Otherwise, $q \notin T \cup F$, and the step function of $M_{(T,F)}$ does nothing. $M_{(T,F)}$ is the partial Boolean mechanism of (T, F) .

NOTE 5. Similar to Observation 1, the profile of a partial Boolean mechanism of (T, F) is the set of all scenarios in which the user's availability vector is in T and the attacker's is in F .

We define a monotonic partial Boolean function as follows:

DEFINITION 13 (MONOTONIC PARTIAL BOOLEAN FUNCTION). A monotonic partial Boolean function is a partial Boolean function that has a monotonic extension.

For example, in Example 1, the mechanism M_{OS}^{ex} is the mechanism of the tuple (T, F) where $T = \{(1, 1)\}$ and $F = \{(0, 0)\}$. This is a monotonic partial Boolean function, as the function $f(x, y) = x \wedge y$ is a monotonic extension of (T, F) .

To prove we can indeed extend a partial Boolean function, we make use of the following observation. We show that if a mechanism M_1 dominates M_2 then the set of user availability vectors in $\Pi(M_1)$ is a subset of the set of user availability vectors in $\Pi(M_2)$, and the same holds for the attacker availability vectors.

OBSERVATION 2. Let M_1 and M_2 be two mechanisms such that $\emptyset \neq \Pi(M_1) \subseteq \Pi(M_2)$. Let T_1 and T_2 be the sets of all user availability vectors and F_1 and F_2 be the sets of all attacker availability vectors in $\Pi(M_1)$ and $\Pi(M_2)$ respectively. Then, $T_1 \subseteq T_2$ and $F_1 \subseteq F_2$.

For example, the mechanisms $M_1 = M_{OS}^{\text{ex}}$ from Example 1, and the mechanism M_2 that requires only the first credential to authenticate, both have non-empty profiles, and $\Pi(M_1) \subseteq \Pi(M_2)$. Then $T_1 = \{(1, 1)\}$, $T_2 = \{(1, 1), (1, 0)\}$, $F_1 = \{(0, 0)\}$ and $F_2 = \{(0, 0), (0, 1)\}$. And it holds that $T_1 \subseteq T_2$ and $F_1 \subseteq F_2$.

We now show that every mechanism is equivalent to a monotonic partial Boolean one.

LEMMA 5. For all mechanisms, there exists an equivalent monotonic partial Boolean mechanism.

Both proofs of Observation 2 and Lemma 5 can be found in Appendix E. Now we show that every maximal mechanism is equivalent to a Boolean mechanism.

THEOREM 2. For all monotonic Boolean mechanisms of non-constant functions, there exists no strictly dominating mechanism.

PROOF. Let f be a non-constant monotonic Boolean function, M_f the mechanism defined by f , and M' a mechanism dominating M_f . We show that $\Pi(M_f) = \Pi(M')$.

By Lemma 5, there exists a monotonic partial Boolean mechanism equivalent to M' . Let (T', F') be the partial Boolean function defining M' . And let T be the set of all user availability vectors in $\Pi(M_f)$ and F be the set of all attacker availability vectors in $\Pi(M_f)$.

First, we show that the profile of M_f is not empty. As f is non-constant, there exists a vector q such that $f(q) = 1$ and a vector q' such that $f(q') = 0$. Consider the scenario σ such that $\sigma_U = q$ and $\sigma_A = q'$. Then, from Observation 1, $\sigma \in \Pi(M_f)$, and thus $\Pi(M_f) \neq \emptyset$.

Then, as $\Pi(M_f) \subseteq \Pi(M')$, from Observation 2, we get that $T \subseteq T'$ and $F \subseteq F'$. However, as f is a Boolean function, $T \cup F = \{0, 1\}^n$ and $T \cap F = \emptyset$. Therefore, $T = T'$ and $F = F'$. And thus, $\Pi(M_f) = \Pi(M')$. \square

Finally, we prove that there exists no hierarchy between monotonic Boolean mechanisms of non-constant functions.

LEMMA 6. Let $f, g : \{0, 1\}^n \rightarrow \{0, 1\}$ be two different non-constant monotonic Boolean functions. Denote by M_f and M_g the monotonic Boolean mechanisms of f and g respectively. If f or g is not constant, then $\Pi(M_f) \neq \Pi(M_g)$.

PROOF. Let $f, g : \{0, 1\}^n \rightarrow \{0, 1\}$ be two different monotonic Boolean functions. There exists a vector $v \in \{0, 1\}^n$ such that $f(v) \neq g(v)$. Assume without loss of generality that $f(v) = 1$ and $g(v) = 0$. As both functions are non-constant, there exists a vector $u \in \{0, 1\}^n$ such that $f(u) = 0$. Consider the scenario $\sigma = (v, u)$. We have $f(v) = 1$ and $f(u) = 0$, and thus $\sigma \in \Pi(M_f)$. We also have $g(v) = 0$, and thus $\sigma \notin \Pi(M_g)$. Therefore, $\Pi(M_f) \neq \Pi(M_g)$. \square

Overall, we showed that for every mechanism that solves the π asynchronous authentication problem, there exists a partial Boolean function that defines it. And there exists a monotonic Boolean mechanism that dominates it. In addition, every monotonic Boolean mechanism of a non-constant function is maximal. Thus concluding that every two monotonic Boolean mechanisms of non-constant functions are either equivalent or incomparable. This shows that in

contrast to the synchronous model [35], there exists no probability-agnostic hierarchy between maximal mechanisms in the asynchronous model.

5 PROBABILISTIC ANALYSIS

Using the properties we found of maximal mechanisms, we present an efficient method for finding mechanisms with approximately maximal success probability.

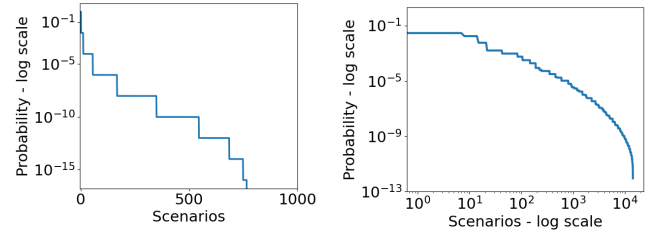
5.1 Preliminaries

As in previous work [18], we derive the probability of scenarios from the credentials' fault probability. Each credential c_i can be in one of the four states, (1) *safe*: Only available to the user, (2) *loss*: Not available to either player, (3) *leak*: Available to both players, or (4) *theft*: Available only to the attacker. Its c_i^S is available to each player accordingly by the scheduler.

Each credential has a probability of being in each of the four states. For example, a complex password that the user memorized has a low probability of being leaked (guessed) but a high probability of being lost (forgotten). While a simple password has a high probability of being leaked but a low probability of being lost. The states of the different credentials are determined by a probability space specified by independent probabilities of each of the credentials. The probabilities of a credential c_i being in each of the four states are denoted by P_i^{safe} , P_i^{loss} , P_i^{leak} , and P_i^{theft} , respectively, $P_i^{safe} + P_i^{loss} + P_i^{leak} + P_i^{theft} = 1$. We assume the mechanism designer estimates the fault probabilities of the credential. But note that this is always the case, even if done implicitly.

The probability of a scenario given the probability vector tuple of each credential is the product of the probabilities of each credential being in the state it is in the scenario. Let M_f be the Boolean mechanism of the function f . The success probability of M_f is the sum of the probabilities of all scenarios in which M_f is successful, denoted by $P^{suc}(M_f)$, and the failure probability is its complement $P^{fail}(M_f) = 1 - P^{suc}(M_f)$. A mechanism is *better* than another if its success probability is higher.

The relation between credentials' fault probabilities and the success probability of a mechanism is not straightforward. For example, consider two credentials c_1 and c_2 . Assume both credentials are prone to loss but not to leakage, with fault probabilities of $P_1^{loss} = P_2^{loss} = 0.1$ and $P_1^{safe} = P_2^{safe} = 0.9$. Then using both credentials in an OR mechanism, where either credential is sufficient for authentication, results in a success probability of 0.99. While using a single credential results in a success probability of 0.9. Therefore, in this case, using both credentials increases the success probability of the mechanism. However, this is not true in general. For example, assume one credential is prone to loss with a fault probability of $P_1^{loss} = 0.1$ and $P_1^{safe} = 0.9$ and the other is prone to leakage with $P_2^{leak} = 0.1$ and $P_2^{safe} = 0.9$. Then using both credentials in an OR mechanism results in a success probability of 0.9. Moreover, any Boolean mechanism using one or both credentials has the same success probability of 0.9. Therefore, in this case, using two credentials does not increase the success probability of the mechanism compared to using a single credential.



(a) $n = 9$: $P_1^{loss} = P_1^{leak} = 0.01$, for $i > 1$ $P_i^{theft} = 0.01$. (b) $n = 7$ with $P^{loss} = 0.05$, $P^{leak} = 0.03$, $P^{theft} = 0.01$.

Figure 1: Probability of scenarios for n credentials and different probabilities.

5.2 Profiles and Scenarios

Given the number of credentials n and the probabilities of states, our goal is to find a mechanism with the highest success probability. Using exhaustive search over the space of all possible mechanisms (equivalently, all monotonic Boolean functions) is infeasible for more than just a few credentials [18]. The number of different Boolean functions with n credentials is the Dedekind number of n [15] which grows super-exponentially and is only known up to $n = 9$ [27]. Therefore, we aim to choose a mechanism that is close to optimal while exploring only a small fraction of the space of all mechanisms. We achieve this by searching by scenarios.

First, we show that there is no need to consider scenarios with no safe credentials.

DEFINITION 14 (VIABLE SCENARIOS). A viable scenario σ is a scenario in which there exists at least one safe credential. A scenario is non-viable if it is not viable. That is, $\sigma_U \leq \sigma_A$.

Maram et al. [35] showed that in the synchronous model, a non-viable scenario is not in the profile of any mechanism. As their model is stronger (has additional assumption on the network), this is also true in our asynchronous model. We also use their following observation.

OBSERVATION 3. The number of viable scenarios for n credentials is $4^n - 3^n$.

Now we bound the number of scenarios that can be added to any partial Boolean mechanism's profile.

OBSERVATION 4. Let (T, F) be a partial Boolean function of n credentials. Let $s = \max(|T|, |F|)$. The maximum number of scenarios that can be added to the profile of the mechanism $M_{(T,F)}$ without contradicting it is $s \cdot (2^n - s) - |\Pi(M_{(T,F)})|$ if $s \geq 2^{n-1}$ and $4^n - 3^n - |\Pi(M_{(T,F)})|$ otherwise.

Both proofs of Observation 3 and Observation 4 are in Appendix F.

To maximize success probability, we limit our search to mechanisms defined by monotonic Boolean functions. We are only interested in credentials with low fault probabilities. Intuitively, scenarios with fewer faults have higher probabilities than those with more faults. So the probability distribution of viable scenarios is highly concentrated on a small number of scenarios. And when some fault probabilities are 0, the number of scenarios with non-zero probability drops even further. Thus, the number of scenarios

Algorithm 2: Scenario-based search algorithm

```

viableScenarios: List of all viable scenarios, sorted by probability in descending order,
calculated based on the fault probabilities of the credentials.
maxSuccessProb  $\leftarrow$  0 // maximal success probability found so far
maxTable  $\leftarrow$  truth table where the all zeroes row is set to 0, the all ones row is set to 1,
and the rest is set to  $\perp$  // truth table with the maximal success probability found so far
 $\delta$ : The precision parameter chosen by the user.
scenarioBasedSearch(truthTable, idx)
/* Input: truthTable- a (possibly partial) monotonic truth table of a mechanism. */
/* Input: idx is the index of the next scenario to consider. */
/* Output: No returned value. The function updates maxSuccessProb and maxTable. */
1 currProfile  $\leftarrow$  truthTable's profile, calculated as described in Note 5
2 successProb  $\leftarrow$  success probability of currProfile
3 if truthTable is complete then
4   if successProb > maxSuccessProb then
5     maxSuccessProb  $\leftarrow$  successProb
6     maxTable  $\leftarrow$  truthTable
7   return
8 possibleScenariosProbabilitiesSorted  $\leftarrow$  sorted array of scenarios probability that can be
added to truthTable and whose index in viableScenarios is higher than idx
9 numPossibleAdditions  $\leftarrow$  maximum number of scenarios that can be added to truthTable
given the table as calculated in Observation 4
10 maxAdditionalProb  $\leftarrow$   $\sum_{i=1}^{numPossibleAdditions} possibleScenariosProbabilitiesSorted(i)$ 
11 if maxAdditionalProb = 0 then // additional scenarios will add 0 to the success probability
12   truthTable  $\leftarrow$  arbitrarily complete truthTable
13   if successProb > maxSuccessProb then
14     maxSuccessProb  $\leftarrow$  successProb
15     maxTable  $\leftarrow$  truthTable
16   return
17 potentialSuccessProb  $\leftarrow$  successProb + maxAdditionalProb // upper bound on success
probability
18 if maxSuccessProb > 0 and maxSuccessProb > potentialSuccessProb -  $\delta$  then
19   return // cannot exceed by over  $\delta$ , prune this branch
20  $\sigma \leftarrow$  viableScenarios(idx) // get next scenario
21 prevUserVal = truthTable( $\sigma_U$ ) // save previous values
22 prevAttVal = truthTable( $\sigma_A$ )
23 if (prevUserVal  $\neq$  0 and prevAttVal  $\neq$  1) and (prevUserVal =  $\perp$  or prevAttVal =  $\perp$ ) then
24   newTable  $\leftarrow$  updateTable(truthTable,  $\sigma$ ) // update table with  $\sigma$  and keep it monotonic
25   scenarioBasedSearch(newTable, idx + 1) // recursive call with the new scenario
26 scenarioBasedSearch(truthTable, idx + 1) // recursive call without the new scenario

```

that our algorithm needs to consider is much smaller than the total number of scenarios. For example, Figure 1a shows the probability distribution of the scenarios for a system with 9 credentials where the first credential might be lost or leaked with probability $p_i^{loss} = p_i^{leak} = 0.01$ and the rest of the credentials might be stolen with probability $p_i^{theft} = 0.01$. The number of scenarios with 9 credentials is $4^9 = 262,144$ with 242,461 viable scenarios, but only 766 have non-zero probability. Similarly, Figure 1b shows the probability distribution of the scenarios for a system with 7 credentials where all fault types are possible with probability $p_i^{loss} = 0.05$, $p_i^{leak} = 0.03$, and $p_i^{theft} = 0.01$ for all credentials. The number of different scenarios with 7 credentials is $4^7 = 16,384$, out of which 14,197 are viable scenarios with positive probability. Yet, over 91% of the probability is concentrated in the first 50 scenarios and over 95% on the first 100 scenarios. Other fault probabilities result in qualitatively similar results. While this case demonstrates a more challenging situation, it still shows that the significant part of the probability is concentrated on a small number of scenarios. Thus, when searching for near-optimal mechanisms, we can neglect the vast majority of scenarios.

5.3 Scenario-Based Search Algorithm

Given a positive fraction $\delta \in \mathcal{N}^+$, our scenario-based search algorithm (Algorithm 2) finds a mechanism whose success probability is at most δ below that of an optimal mechanism. It takes advantage of three key observations: (1) The number of scenarios that might be

added to a profile of a partial Boolean mechanism is bounded (Observation 4); (2) non-viable scenarios can be ignored [35]; and (3) the probability of different scenarios drops quickly (Figure 1).

Instead of searching all possible mechanisms to find the optimal one, we build the truth table (and thus mechanism) incrementally: Given the number of credentials n , we start with the empty partial truth table of n variables of the function (total of 2^n rows) and add rows. By Lemma 5, for all mechanisms M there exists a monotonic partial Boolean function (T, F) such that M is equivalent to the mechanism defined by (T, F) . The monotonic partial truth table corresponding to (T, F) has the value 1 for all vectors in T , 0 for all vectors in F , and the rest of the vectors $v \notin T \cup F$ are not set yet (denoted by \perp). Changing a value of a row in the truth table from \perp to 0 is equivalent to adding the vector to F and changing a value from \perp to 1 is equivalent to adding the vector to T .

By Theorem 1 every mechanism is dominated by a Boolean mechanism. Because every maximal mechanism is a monotonic Boolean mechanism (Theorem 2), and every monotonic Boolean function yields a maximal mechanism (Lemma 6), we extend the monotonic partial Boolean function that defines the mechanism until it is full (representing a Boolean function).

Note that if the value of the all zeroes row is set to 1 in the truth table, from monotonicity, we get the constant function $f(x) = 1$: As for all $x \in \{0, 1\}^n$ we have that $x \geq 0^n$, then to ensure monotonicity, we must set $f(x) = 1$ for all $x \in \{0, 1\}^n$. Similarly, if the value of the all ones row is set to 0, we get the constant function $f(x) = 0$.

It is easy to confirm that the profile of a Boolean mechanism of a constant function is empty: Let M_F and M_T be the mechanisms defined by the constant Boolean functions $\forall v, F(v) = 0$, and $T(v) = 1$. Let $\sigma = (\sigma_U, \sigma_A)$ be a scenario. Because $\forall v, F(v) = 0$, we get that $F(\sigma_U) \wedge \neg F(\sigma_A) = 0$, thus M_F fails. Similarly, we have $T(\sigma_U) \wedge \neg T(\sigma_A) = 0$. Therefore, both mechanisms have empty profile. So we consider only non-constant monotonic Boolean functions and set the value of the all zeroes row to 0 and the value of the all ones row to 1.

Equipped with these insights, we are ready to present the scenario-based search (Algorithm 2). Given the credentials' probability vectors, we calculate the probability of each viable scenario, sort it and save it in a global variable *viableScenarios*. Then we progress greedily, from scenarios with the highest to the lowest probabilities. For each scenario, there are two options: either include it in the profile or not. We refer to *truthTable* as a dictionary mapping an availability vector to 0, 1, or \perp . We denote by *truthTable*(σ_U) and *truthTable*(σ_A) the value of the user and the attacker's availability vectors σ_U and σ_A in *truthTable* respectively. Similarly, we denote by *viableScenarios*(*idx*) the scenario at index *idx* in the sorted list of viable scenarios.

In each step, the algorithm computes the success probability of the current partial truth table (lines 1-2). It then checks if the truth table is full (line 3). If so, it means we reached a Boolean function that cannot be extended. The algorithm checks if the success probability of the mechanism (truth table) is higher than the best found so far (line 4). If so, it updates the best success probability and the corresponding global best truth table (lines 5-6), then it returns.

Otherwise, the table can be extended. The algorithm checks if it can add any scenarios with positive probability to the current

truth table (line 11). If not, then any additional scenarios would contribute 0 to the success probability, regardless of the exact scenarios chosen. Thus, it completes the truth table arbitrarily by iteratively adding scenarios from those left (with a higher index than idx in $viableScenarios$) while making sure the result is monotonic (line 12), checks if the success probability of the resulting mechanism is higher than the global best found so far (line 13), and updates the global best success probability and the corresponding global best truth table accordingly (lines 14-15), then it returns. Note that this covers the case where no scenarios are left to add.

Then, the algorithm checks if it is beneficial to continue extending the current table. It does so by finding an upper bound on the additional success probability for the current branch by considering the next $numPossibleAdditions$ scenarios in the sorted list and summing their probabilities (line 10) where $numPossibleAdditions$ (line 9) is the maximum number of scenarios that may be added to the current truth table without contradicting it (Observation 4). If the potential best solution is not better by at least δ than the current best (line 18), we prune this branch and do not consider any of the mechanisms it results in (line 19). This takes advantage of the fact that the probability of different scenarios drops exponentially fast (Figures 1a and 1b) and is a key step in ensuring the algorithm’s efficiency.

If none of the stop conditions hold, the algorithm continues by exploring the next scenario (line 20). For the recursive exploration, given a scenario, we have two options: (1) Include the scenario in the truth table and recursively call the algorithm (only if it does not contradict the current truth table and is not already in the profile) (line 25). Or (2) Exclude this scenario and recursively call the algorithm with the next scenario (line 26).

Given a scenario σ and a partially filled truth table, we check whether the scenario contradicts the truth table and whether it is already in the profile (line 23). We do so by checking if the value of the user’s availability vector σ_U is not set to 0 (either 1 or \perp) and the value of the attacker’s availability vector σ_A is not set to 1 (either 0 or \perp). If so, the scenario does not contradict the truth table. And checking that either the user’s availability vector or the attacker’s availability vector is not set yet (has a value \perp). If so, then it is not in the profile. If both conditions hold, we include the scenario in the truth table and recursively call the algorithm (line 25).

To include a scenario σ in the profile, we create an updated truth table using the function `updateTable(truthTable, σ)` (line 24). The function sets the entry corresponding to σ_U to 1 and the entry σ_A to 0. Then, to have the truth table represent a monotonic Boolean function, `updateTable(\cdot)` updates it by setting all the entries $x \in \{0, 1\}^n$ such that $x > \sigma_U$ to 1. And setting all entries $y \in \{0, 1\}^n$ such that $y < \sigma_A$ to 0.

The algorithm’s result is the mechanism saved in the $maxTable$ and the corresponding success probability $maxSuccessProb$.

5.4 Algorithm Correctness

We now show that our scenario-based search (Algorithm 2) finds a mechanism that is δ close to the optimal mechanism. First, we note that after each update to the partial monotonic truth table, it remains a (possibly partial) monotonic truth table—the above update does not contradict previous ones (Appendix G.1). Next, we

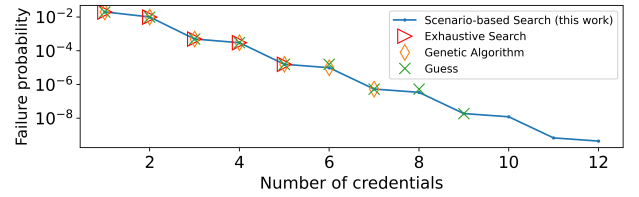


Figure 2: Failure probability vs. credentials number. Comparing our algorithm to previous methods: exhaustive search, genetic algorithm, and guessing a symmetric mechanism.

show that the algorithm always stops and that when it returns, the truth table contains a valid complete monotonic truth table whose success probability is at most δ less than the optimal mechanism.

The algorithm explores adding scenarios to the truth table incrementally, in the worst case it may explore all viable scenarios. As the number of viable scenarios is finite and bounded (Observation 3), the algorithm always stops.

The algorithm stops only when one of the three stopping conditions is met. In the first case (line 3), the truth table is complete, and the algorithm updates $maxTable$ to this complete table. In the second case (line 11), the algorithm finds that all scenarios that can be added to the current truth table will not improve the success probability. So it completes the truth table arbitrarily, and $maxTable$ gets the completed table. Finally, in the third case (line 18), any possible additions do not result in a mechanism that is at least δ better so the algorithm prunes the branch and returns, without updating $maxTable$. However, this condition is met only if $maxSuccessProb$ is greater than 0 i.e., if the algorithm found a solution earlier. Therefore, the algorithm must update a full truth table at least once, and it is always a valid monotonic truth table that has a success probability at most δ less than the optimal mechanism. The proof is in Appendix G.1.

5.5 Algorithm Complexity

We first present an upper bound on the complexity of the algorithm, followed by a discussion of when and why the actual complexity is much lower. Then we evaluate the complexity of the algorithm empirically (§5.5.2), by measuring the runtime of our algorithm for different numbers of credentials and fault probabilities.

5.5.1 Bound. To give a loose upper bound on the algorithm’s time complexity, we note that each recursive call’s complexity is bounded by $O((4^n - 3^n) \cdot n)$. And in the worst case, the recursion depth reaches $O(2^{4^n - 3^n})$. The overall bound is thus $O(2^{4^n - 3^n} \cdot (4^n - 3^n) \cdot n)$. We defer a detailed analysis to Appendix G.2.1.

This analysis assumes the worst case for every step of the algorithm. However, in practice, many steps’ worst case cannot happen simultaneously. Thus, the actual complexity of the algorithm is much lower, depending on the number of credentials, their probabilities, and the parameter δ .

For example, while an upper bound on the recursion depth is given by 2 to the power of the number of viable scenarios, the actual depth explored is much smaller. This is due to multiple factors, for example (1) the exponential drop in the probability of different scenarios, combined with the fact that the algorithm prunes branches with negligible advantage; (2) the fact that the number of scenarios

that can be added to a profile of a partial Boolean mechanism is limited (Observation 4) in a non-trivial way depending on which do result in a monotonic mechanism; and (3) each scenario adds not a single row to the truth table, but possibly many rows for keeping monotonicity (depending on the specific scenario and the current truth table). While all those factors contribute to the algorithm’s efficiency, it remains an open question whether the algorithm’s exact complexity can be calculated theoretically (cf. the lack of a closed form expression for the number of monotonic Boolean functions [15]).

5.5.2 Empirical Complexity. As the problem of finding the exact complexity of the algorithm remains open, we evaluate the algorithm’s complexity by measuring its runtime as a function of the number of credentials and their different fault probabilities. Figure 3 shows the runtime of the algorithm for different numbers of credentials and fault probabilities using a logarithmic Y scale.

We consider two different categories of results: One where the algorithm’s runtime grows exponentially with the number of credentials, and one where it grows super-exponentially.

Exponential Growth: When one credential can suffer from up to a single type of fault and the rest can have up to two types of faults, or when all credentials can suffer from all types of faults with low probability, the algorithm’s runtime grows exponentially ($O(4^n)$) with the number of credentials (all fits with $R^2 > 0.97$, exact values are in Appendix G.2.2). For example, when all credentials can suffer from loss with $p^{loss} = 0.01$, $p^{leak} = p^{theft} = 0$ (only loss in Figure 3), or when 2 credentials can be easily lost, but not leaked or stolen, with $i \in \{1, 2\}$, $p_i^{loss} = 0.3$, $p_i^{leak} = p_i^{theft} = 0$ and the rest can be either lost or leaked with $j > 2$, $p_j^{loss} = p_j^{leak} = 0.01$, $p_j^{theft} = 0$. (2 easily-to-lose in Figure 3).

We observe a similar trend when each credential can have all three types of faults, where at least two with low fault probabilities, $p^{loss} = 0.01$, $p^{leak} = p^{theft} = 0.001$ (loss, leak, theft in Figure 3).

In all such cases, our scenario-based search runtime grows exponentially slower (better) than the exhaustive search which requires at least $O(D(n) \cdot 4^n)$ operations, where $D(n)$ is the number of monotonic Boolean functions of n variables. Notice that this improvement was crucial for enabling the forthcoming case studies.

Super-Exponential Growth: But in some cases complexity is worse. When all credentials can suffer from two or more types of faults with a high probability, the algorithm’s runtime grows super-exponentially with the number of credentials. E.g., if all three fault types have high probabilities, say, $p^{loss} = 0.1$, $p^{leak} = 0.3$, $p^{theft} = 0.4$, the algorithm’s runtime grows too rapidly to obtain sufficient data points for regression analysis.

5.6 Case Studies

The algorithm allows designers to choose approximately-optimal mechanisms given their credentials’ fault probabilities. As the actual fault probabilities of credentials are application-specific and not publicly available, a user or a company can use the algorithm to find the best mechanism for their actual case. The ability to study mechanisms with a larger number of credentials compared to previous work allows taking advantage of elaborate schemes. Our focus is on providing a tool for finding approximately-optimal

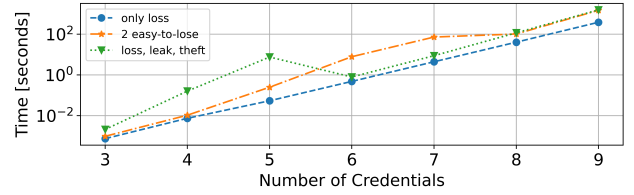


Figure 3: Runtime of the algorithm as a function of the number of credentials for different fault probabilities with $\delta = 10^{-5}$.

mechanisms rather than on specific mechanisms. We use concrete values to evaluate the algorithm’s efficacy and accuracy (§5.6.1) and propose improvements for cryptocurrency wallets (§5.6.2) and for online authentication (§5.6.3).

5.6.1 Algorithm evaluation: We demonstrate the efficacy and accuracy of our method by comparing the results to the optimal mechanisms found using exhaustive search, a heuristic genetic algorithm [18], and guessing a symmetric k/n mechanism [18]. As expected, the number of credentials has a major effect on wallet security. Figure 2 shows the security of the optimal mechanism for different numbers of credentials when the first credential can be lost or leaked, but not stolen ($p^{loss} = p^{leak} = 0.01$), and the rest of the credentials can only be stolen ($p^{theft} = 0.01$). We use a parameter δ ranging from 10^{-5} for smaller numbers of credentials to 10^{-6} for larger numbers. Our algorithm efficiently finds the same optimal mechanism as the exhaustive search and is able to find an approximately optimal mechanism for up to 12 credentials. In several cases, it finds better mechanisms than the ones found by guessing a symmetric mechanism, e.g., Figure 2 with 6 or 8 credentials.

5.6.2 Cryptocurrency wallets. Typically, cryptocurrency wallets use a single private key or mnemonic [14, 38] or multiple such credentials with so-called multisig wallets or using threshold signatures [12, 55], e.g., 2 out of 2 or 2 out of 3. These mechanisms rely on users’ ability to securely store their credentials, and thus require low credential fault probabilities. While increasing the number of strong keys improves security [18], storing strong keys is challenging [6, 34]. However, it is possibly easier to store credentials with high loss probability, e.g., a hard password committed to memory that the user might forget but is hard to guess.

By exploring the larger design space with many credentials, we find wallets can take advantage of weak credentials to improve security. For example, in the case of a 2/2 multisig mechanism, using two additional weak credentials reduces the failure probability by an order of magnitude from $2 \cdot 10^{-2}$ to $6 \cdot 10^{-3}$. Figure 4 shows the failure probability for different numbers of credentials when all credentials can be lost or leaked, but not stolen ($p^{loss} = p^{leak} = 0.01$), compared to the security when using the same credentials but adding weak credentials that can only be lost with higher probability ($p^{loss} = 0.3$). Here we used the parameter $\delta = 10^{-6}$. To the best of our knowledge, there are no public user studies or statistics on credential fault probabilities. However, we considered different combinations of possible fault probabilities. The results are qualitatively similar. We find that for n regular credentials and k easy-to-lose credentials, the

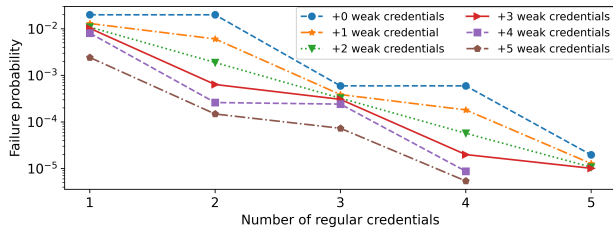


Figure 4: Failure probability harnessing easy-to-lose credentials. Weak: $p^{leak} = 0, p^{loss} = 0.3$, **regular:** $p^{leak} = p^{loss} = 0.01$.

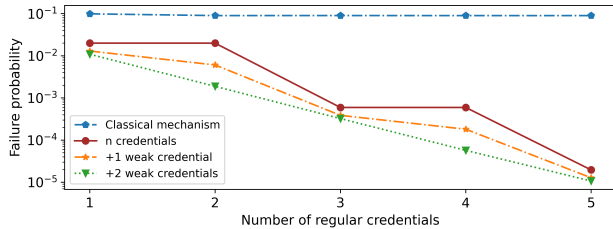


Figure 5: Failure probability harnessing easy-to-leak credentials. Weak: $p^{leak} = 0.3, p^{loss} = 0$, **regular:** $p^{leak} = p^{loss} = 0.01$.

optimal mechanism design is to require any of the k easy-to-lose credentials or any $\lfloor n/2 \rfloor + 1$ of the n regular credentials.

5.6.3 Security questions. Until a few years ago, large email providers like Google, Microsoft, and Yahoo! employed security questions (like “name of first pet”) for password reset in online services [6]. But using security questions in this simple manner famously reduces security [45, 51]. The monotonic Boolean function representing such a mechanism is: require all answers to the security questions or the password. Or in general, require all answers to the security questions or any k out of n of the regular credentials (e.g., password, fingerprint, or SMS code). We call this the classical mechanism.

The main issue is that the answers to the security questions are often easy to guess or to find online, that is, they are easy-to-leak credentials. This implies that, with a fairly high probability, an attacker can gain access to the account by answering the security questions. And, indeed, most online services have stopped using security questions [8] (with a few exceptions, e.g., [4, 29]).

Perhaps surprisingly, we find that a well-designed mechanism can actually take advantage of easy-to-leak credentials. We used the algorithm to find an approximately optimal mechanism and compared it against the classical mechanism. Figure 5 shows the failure probability of the mechanism for different numbers of credentials when all credentials can be lost or leaked ($p^{loss} = p^{leak} = 0.01$), compared to the security when using the same credentials but adding weak credentials that can only be leaked with higher probability ($p^{leak} = 0.3$). Just two easy-to-leak keys reduce the failure probability by about an order of magnitude. Again, we experimented with different combinations of possible fault probabilities and the results were qualitatively similar. The resultant mechanism requires answering the security questions in addition to having any $\lfloor n/2 \rfloor$ out of the n regular credentials.

6 CONCLUSION

We formalize the common asynchronous authentication problem, reconciling the apparent tension between asynchrony and cryptographic security. We show every mechanism is dominated by a non-trivial monotonic Boolean mechanism, which itself is not strictly dominated. So in every pair of distinct such mechanisms, neither strictly dominates the other. We present a practical algorithm for finding approximately optimal mechanisms, given the credential fault probabilities. This highlights the need for user studies to quantify those probabilities.

Our algorithm reveals surprising results: Accurately incorporating weak credentials improves mechanisms’ security by orders of magnitude. One case study shows that a user designing her cryptocurrency wallet can benefit from incorporating with her carefully-guarded credentials a few additional ones that are perhaps easy to lose. This is achieved by a mechanism that requires any of the easy-to-lose credentials or any $\lfloor n/2 \rfloor + 1$ out of the n regular credentials. Another case study shows that using security questions (that are easy-to-leak) in addition to regular credentials can significantly improve security if used wisely, by requiring any $\lfloor n/2 \rfloor$ out of the n regular credentials and the answers to the security questions. More generally, both individuals and companies can use these results directly to rigorously design authentication mechanisms addressing the demands of current and forthcoming challenges.

ACKNOWLEDGMENTS

This work was supported in part by Avalanche Foundation and by IC3.

REFERENCES

- [1] Kumar Abhishek, Sahana Roshan, Prabhat Kumar, and Rajeev Ranjan. 2013. A Comprehensive Study on Multifactor Authentication Schemes. In *Advances in Computing and Information Technology*, Natarajan Meghanathan, Dhinakaran Nagamalai, and Nabendu Chaki (Eds.). Springer Berlin Heidelberg.
- [2] Shannon Appelcline. 2021. Using Timelocks to Protect Digital Assets. <https://github.com/BlockchainCommons/SmartCustody/blob/master/Docs/Timelocks.md>. Accessed, Dec 2023.
- [3] Argent. 2021. Argent Smart Wallet Specification. <https://github.com/argentlabs/argent-contracts/blob/develop/specifications/specifications.pdf>. Accessed, Dec 2023.
- [4] MT Bank. 2024. MT Bank Reset Passcode. <http://tinyurl.com/mtd48pm9>. Accessed, Jan 2024.
- [5] Simon Barber, Xavier Boyen, Elaine Shi, and Ersin Uzun. 2012. Bitter to Better – How to Make Bitcoin a Better Currency. In *Financial Cryptography and Data Security*, Angelos D. Keromytis (Ed.). Springer Berlin Heidelberg.
- [6] 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 Symposium on Security and Privacy*. <https://doi.org/10.1109/SP.2012.44>
- [7] 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 *2015 IEEE Symposium on Security and Privacy*. <https://doi.org/10.1109/SP.2015.14>
- [8] Joseph Bonneau and Sören Preibusch. 2010. The Password Thicket: Technical and Market Failures in Human Authentication on the Web. In *WEIS*.
- [9] Endre Boros, Vladimir Gurvich, Peter L Hammer, Toshihide Ibaraki, and Alexander Kogan. 1995. Decomposability of partially defined Boolean functions. *Discrete Applied Mathematics* 62, 1-3 (1995).
- [10] Leon Bošnjak and Bostjan Brumen. 2019. Rejecting the death of passwords: Advice for the future. *Computer Science and Information Systems* 16 (01 2019). <https://doi.org/10.2298/CSIS180328016B>
- [11] Michael Burrows, Martin Abadi, and Roger Needham. 1990. A Logic of Authentication. *ACM Trans. Comput. Syst.* 8, 1 (1990). <https://doi.org/10.1145/77648.77649>
- [12] Camino. 2024. Create and Edit Multisig Wallets. <https://docs.camino.network/guides/multisig-wallets/create-multisig>. Accessed, Jan 2024.

- [13] Ran Canetti. 2001. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*. IEEE.
- [14] Coinbase. 2024. Coinbase Wallet. <https://www.coinbase.com/wallet>. Accessed, Jan 2024.
- [15] R. Dedekind. 1897. *Über Zerlegungen von Zahlen Durch Ihre Grössten Gemeinsamen Theiler*. Vieweg+Teubner Verlag, Wiesbaden. https://doi.org/10.1007/978-3-663-07224-9_1
- [16] Cynthia Dwork and Aaron Roth. 2014. The Algorithmic Foundations of Differential Privacy. *Found. Trends Theor. Comput. Sci.* 9, 3–4 (Aug 2014). <https://doi.org/10.1561/04000000042>
- [17] Jonathan Eaton. 2011. The Political Significance of the Imperial Watchword in the early Empire. *Greece and Rome* 58, 1 (2011). <https://doi.org/10.1017/S0017383510000525>
- [18] Ittay Eyal. 2022. On cryptocurrency wallet design. In *3rd International Conference on Blockchain Economics, Security and Protocols (Tokenomics 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik.
- [19] Federal Trade Commission. 2022. Consumer sentinel network data book 2021.
- [20] Joan Feigenbaum. 1992. Overview of interactive proof systems and zero-knowledge. *Contemporary Cryptology: The Science of Information Integrity* (1992).
- [21] Google. 2024. How Google uses cookies. <https://policies.google.com/technologies/cookies?hl=en-GB>. Accessed, May 2024.
- [22] Google. 2024. How Google uses location information. <https://policies.google.com/technologies/location-data?hl=en-GB>. Accessed, May 2024.
- [23] Google. 2024. Respond to security alerts. <https://support.google.com/accounts/answer/2590353?hl=en>. Accessed, Jan 2024.
- [24] United States government. 2024. Authentication methods. <https://www.login.gov/help/get-started/authentication-methods/>. Accessed, May 2024.
- [25] Joseph Y. Halpern and Yoram Moses. 1990. Knowledge and Common Knowledge in a Distributed Environment. *J. ACM* 37, 3 (Jul 1990). <https://doi.org/10.1145/79147.79161>
- [26] Sven Hammann, Saša Radomirović, Ralf Sasse, and David Basin. 2019. User Account Access Graphs. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (CCS '19)*. Association for Computing Machinery, New York, NY, USA. <https://doi.org/10.1145/3319535.3354193>
- [27] Lennart Van Hirtum, Patrick De Causmaecker, Jens Goemaere, Tobias Kenter, Heinrich Riebler, Michael Lass, and Christian Plessl. 2023. A computation of D(9) using FPGA Supercomputing. arXiv:2304.03039 [cs.DM]
- [28] Dennis Hofheinz, Dominique Unruh, and Jörn Müller-Quade. 2013. Polynomial runtime and composability. *Journal of Cryptology* 26 (2013).
- [29] Leumi International. 2024. Leumi International - Security Questions. <https://english.leumi.co.il/Articles/32891/>. Accessed, Jan 2024.
- [30] Prashant Jha. April 2022. The aftermath of Axie Infinity’s \$650M Ronin Bridge hack. *Cointelegraph* (April 2022). <https://cointelegraph.com/news/the-aftermath-of-axie-infinity-s-650m-ronin-bridge-hack>.
- [31] Leslie Lamport. 1981. Password Authentication with Insecure Communication. *Commun. ACM* 24, 11 (nov 1981). <https://doi.org/10.1145/358790.358797>
- [32] Ledger. 2024. Ledger Hardware Wallet. <https://www.ledger.com/>. Accessed, Jan 2024.
- [33] Ninghui Li, Benjamin N. Groszof, and Joan Feigenbaum. 2003. Delegation Logic: A Logic-Based Approach to Distributed Authorization. *ACM Trans. Inf. Syst. Secur.* 6, 1 (feb 2003). <https://doi.org/10.1145/605434.605438>
- [34] 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*, Paper 2022/075. <https://eprint.iacr.org/2022/075> <https://eprint.iacr.org/2022/075>
- [35] Deepak Maram, Mahimna Kelkar, and Ittay Eyal. 2022. Interactive Authentication. *Cryptology ePrint Archive*, Paper 2022/1682. <https://eprint.iacr.org/2022/1682> <https://eprint.iacr.org/2022/1682>
- [36] Griffin Messhane. 2022. What Is a Multisig Wallet? <https://www.coindesk.com/learn/what-is-a-multisig-wallet/>. Accessed, Dec 2023.
- [37] John Mecke. 2019. 5 Stories About People Who Lost Their Bitcoin. <https://john-mecke.medium.com/5-stories-about-people-who-lost-their-bitcoin-cdaae329468>. Accessed, Jan 2024.
- [38] Metamask. 2024. Get started with MetaMask Portfolio. <https://metamask.io/>. Accessed, Jan 2024.
- [39] Robert Morris and Ken Thompson. 2002. Password Security: A Case History. *Commun. ACM* 22 (05 2002). <https://doi.org/10.1145/359168.359172>
- [40] Scott Nevil. 2023. Bitcoin safe storage - cold wallet. <https://www.investopedia.com/news/bitcoin-safe-storage-cold-wallet/>. Accessed, Dec 2023.
- [41] Soumya Prakash Ota, Subhrakanta Panda, Maanak Gupta, and Chittaranjan Hota. 2023. A Systematic Survey of Multi-Factor Authentication for Cloud Infrastructure. *Future Internet* 15, 4 (2023). <https://doi.org/10.3390/fi15040146>
- [42] Charles P. Pfleeger and Shari Lawrence Pfleeger. 2012. *Security in Computing, 4th Edition*. Prentice Hall.
- [43] Youer Pu, Lorenzo Alvisi, and Ittay Eyal. 2022. Safe Permissionless Consensus. In *36th International Symposium on Distributed Computing*.
- [44] Youer Pu, Ali Farahbakhsh, Lorenzo Alvisi, and Ittay Eyal. 2023. Gorilla: Safe Permissionless Byzantine Consensus. In *37th International Symposium on Distributed Computing*.
- [45] Ariel Rabkin. 2008. Personal knowledge questions for fallback authentication: Security questions in the era of Facebook. In *Proceedings of the 4th Symposium on Usable Privacy and Security*.
- [46] BCC Research. 2022. Non-Fungible Tokens (NFT): Global Market. <https://www.bccresearch.com/market-research/information-technology/nft-market.html>. Accessed: June 2023.
- [47] Grand View Research. 2023. Cryptocurrency Market Size, Share and Growth Report, 2030. <https://www.grandviewresearch.com/industry-analysis/cryptocurrency-market-report>. Accessed, Dec 2023.
- [48] Keunwoo Rhee, Jin Kwak, Seungjoo Kim, and Dongho Won. 2005. Challenge-response based RFID authentication protocol for distributed database environment. In *Security in Pervasive Computing: Second International Conference, SPC 2005, Boppard, Germany, April 6–8, 2005. Proceedings 2*. Springer.
- [49] R. L. Rivest, A. Shamir, and L. Adleman. 1978. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Commun. ACM* 21, 2 (feb 1978). <https://doi.org/10.1145/359340.359342>
- [50] Norman Field (HYPR) Salah Machani (RSA Security). 2022. *FIDO Alliance White Paper: Choosing FIDO Authenticators for Enterprise Use Cases*. Technical Report. FIDO Alliance.
- [51] Stuart Schechter, AJ Bernheim Brush, and Serge Egelman. 2009. It’s no secret: measuring the security and reliability of authentication via secret questions. In *2009 30th IEEE symposium on security and privacy*. IEEE.
- [52] Fred B. Schneider. 1990. Implementing Fault-Tolerant Services Using the State Machine Approach: A Tutorial. *ACM Comput. Surv.* 22, 4 (dec 1990). <https://doi.org/10.1145/98163.98167>
- [53] Bruce Schneier. 1996. *Applied cryptography: protocols, algorithms, and source code in C* (2nd ed.). Wiley, New York.
- [54] Richard E Smith. 2001. *Authentication: from passwords to public keys*. Addison-Wesley Longman Publishing Co., Inc.
- [55] Speior. 2023. Advanced MPC Wallets for Digital Assets and Custody Infrastructure. <https://sepior.com/>. Accessed, Jan 2024.
- [56] William Stallings. 2003. *Cryptography and network security - principles and practice* (3. ed.). Prentice Hall.
- [57] William. Stallings and Lawrie. Brown. 2012. *Computer security : principles and practice / William Stallings, Lawrie Brown ; with contributions by Mick Bauer, Michael Howard* (2nd ed. ed.). Pearson Boston.
- [58] Jennifer G. Steiner, B. Clifford Neuman, and Jeffrey I. Schiller. 1988. Kerberos: An Authentication Service for Open Network Systems. In *USENIX Winter*.
- [59] Andrew S Tanenbaum. 2007. *Distributed systems principles and paradigms*.
- [60] Chainalysis Team. 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/>.
- [61] Trezor. 2024. Trezor Hardware Wallet. <https://trezor.io/>. Accessed, Jan 2024.
- [62] Arti Vaish, Anand Sharma, and Anshu Sharma. 2020. Review Reports on User Authentication Methods in Cyber Security. *WSEAS TRANSACTIONS ON COMMUNICATIONS* 19 (10 2020). <https://doi.org/10.37394/23204.2020.19.17>
- [63] Maarten Van Steen and Andrew S Tanenbaum. 2017. *Distributed systems*. Maarten van Steen Leiden, The Netherlands.
- [64] Ignacio Velásquez, Angélica Caro, and Alfonso Rodríguez. 2018. Authentication schemes and methods: A systematic literature review. *Information and Software Technology* 94 (2018). <https://doi.org/10.1016/j.infsof.2017.09.012>
- [65] Jiaping Wang and Hao Wang. 2019. Monoxide: Scale out Blockchains with Asynchronous Consensus Zones. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. USENIX Association, Boston, MA. <https://www.usenix.org/conference/nsdi19/presentation/wang-jiaping>
- [66] Yong Zhu, Tieniu Tan, and Yunhong Wang. 2000. Biometric personal identification based on iris patterns. In *Proceedings 15th International Conference on Pattern Recognition. ICPR-2000*, Vol. 2. <https://doi.org/10.1109/ICPR.2000.906197>
- [67] Verena Zimmermann, Nina Gerber, Peter Mayer, Marius Kleboth, Alexandra von Preuschen, and Konstantin Schmidt. 2019-11-11. Keep on rating – on the systematic rating and comparison of authentication schemes. *Information and computer security*, 27, 5 (2019-11-11).
- [68] Zuriati Ahmad Zukarnain, Amgad Muneer, and Mohd Khairulnauar Ab Aziz. 2022. Authentication Securing Methods for Mobile Identity: Issues, Solutions and Challenges. *Symmetry* 14, 4 (2022). <https://doi.org/10.3390/sym14040821>

A SECURITY PARAMETERS AND MECHANISMS FAMILIES

A.1 Mechanisms With Security Parameters

DEFINITION 15 (AUTHENTICATION MECHANISM). An authentication mechanism M^λ is a finite deterministic automaton, parametrized

by the security parameter λ and specified by two functions, $gen_{M^\lambda}(\cdot)$ and $step_{M^\lambda}(\cdot)$:

- $gen_{M^\lambda}(n)$, where n is the number of credentials (Note that the security parameter is given implicitly with M^λ), is a secure credential generator. And
- $step_{M^\lambda}(msg, i)$, where msg is the message the mechanism received (perhaps no message, represented by $msg = \perp$), sent from player i , is a function that updates the state of the mechanism and returns a pair $(msgs_0, msgs_1)$ of sets of messages (maybe empty) to send to the players by their identifiers. It can access and update the mechanism's state and the set of credentials public part $\{c_i^P\}_{i=1}^n$.

A player is defined by its strategy, a function that defines its behavior. Formally,

DEFINITION 16 (STRATEGY). A strategy S^λ of a player p is a function, parametrized by the security parameter λ , that takes a message from the mechanism as input (which may be empty, denoted by \perp), and has access to the player's state and credentials C_p^σ . It updates the player's state and returns a set of messages to be sent back to the mechanism.

A.2 Execution

We now describe the execution of the system. It consists of two parts: *setup* (Setup function, Algorithm 3) and a *main loop* (Execute function, Algorithm 4). The setup generates a set of n credentials $\{c_i = (c_i^P, c_i^S)\}_{i=1}^n$ using the function $gen_{M^\lambda}(n)$ (line 1). The credential generation function $gen_{M^\lambda}(n)$ draws randomness from the random tape v and returns a set of n credentials. Then the setup function assigns each player credentials according to the scenario σ (lines 2-3), and the mechanism receives all the credentials' public parts (line 4). Then it assigns an identifier to each player: $\gamma_{ID} \in \{0, 1\}$ to the user (line 5) and $(1 - \gamma_{ID})$ to the attacker (line 6). Whenever a random coin is used by M^λ or the players, the result is the next bit of v .

Once the setup is complete, the main loop begins (line 2). From this point onwards in the execution, both players $p \in \{U, A\}$ can send messages to the mechanism, each based on her strategy S_p^λ and the set of credentials available to her C_p^σ .

Time progresses in discrete steps t , accessible to the mechanism and players. The communication network is reliable but asynchronous. Messages arrive eventually, but there is no bound on the time after they were sent and no constraints on the arrival order. This is implemented as follows. The scheduler maintains three sets of pending messages, one for each entity identifier $e \in \{0, 1, M^\lambda\}$, denoted by $Pending_e$. The scheduler is parametrized by an ordering function γ_{ord} .

In each step, it uses γ_{ord} to choose a subset of messages (maybe empty) from each of the pending message sets, and returns them as an ordered list (line 6). The scheduler removes the chosen messages from the pending messages sets (line 7). The function γ_{ord} gets as input the identifier of the entity $e \in \{0, 1, M^\lambda\}$ that the messages are sent to. It also has access to the state of the execution and a separate scheduler random tape v_γ . However, γ_{ord} does not have access to the messages themselves, the security parameter nor to the entities' random tape v . We use an ordering function that's

Algorithm 3: Setup of the system, with mechanism M^λ in scenario σ

```

Setup( $M^\lambda, \sigma, S_U^\lambda, S_A^\lambda, \gamma, v$ )
1  $C \leftarrow gen_{M^\lambda}(n)$  // Generating the set of credentials  $C = \{c_i = (c_i^P, c_i^S)\}_{i=1}^n$ 
2  $C_U^\sigma \leftarrow \{c_i^S \mid \sigma_{U_i} = 1\}$  // Assigning the credentials' secret parts to the user
3  $C_A^\sigma \leftarrow \{c_i^S \mid \sigma_{A_i} = 1\}$  // Assigning the credentials' secret parts to the attacker
4  $C_M^\sigma \leftarrow \{c_i^P \mid i \in \{1, \dots, n\}\}$  // Assigning the credentials' public parts to the mechanism
5  $S_{\gamma_{ID}}, C_{\gamma_{ID}} \leftarrow S_U^\lambda, C_U^\sigma$  // Assigning  $\gamma_{ID}$  to the user
6  $S_{1-\gamma_{ID}}, C_{1-\gamma_{ID}} \leftarrow S_A^\lambda, C_A^\sigma$  // Assigning  $1 - \gamma_{ID}$  to the attacker
7 return  $M^\lambda, S_0, C_0, S_1, C_1, v$ 

```

oblivious to the messages content and the security parameter to model a system in which, for a sufficiently large security parameter, messages eventually arrive during the execution. The ordering function's lack of access to the messages and the security parameter ensures that it cannot delay messages until after the execution ends for all λ .

Using two separate random tapes, one for the scheduler and another for all other entities in the system, is only for presentation purpose. This is equivalent to using a single global random tape. Because there exists a mapping between the two, e.g., given a global random tape, we map it into two separate tapes such that the elements in the odd indices are mapped to the scheduler tape and the elements in the even indices are mapped to the other entities' tape.

Each player $p \in \{U, A\}$ receives the messages chosen by the ordering function γ_{ord} and sends messages to the mechanism according to her strategy S_p^λ (lines 9-12). Messages sent by the players are added to the mechanism's pending messages set $Pending_M$. Similarly, the mechanism receives the messages chosen by γ_{ord} . After every message msg it receives, the mechanism updates its state using its function $step_{M^\lambda}(msg)$ and returns messages to add to the players' pending messages sets $Pending_0, Pending_1$ (line 14). It then checks if the mechanism has decided which one of the players is recognized as the user by checking its variable $decide_{M^\lambda}$ (lines 17-18). Once the mechanism reaches a decision, the execution ends and the player with the matching index (either 0 or 1) wins. The tuple $(\gamma_{ID}, \gamma_{ord}, v_\gamma)$ thus defines the scheduler's behavior. And an execution E^λ (Algorithm 5) is thus defined by its parameter tuple $(M^\lambda, \sigma, S_U^\lambda, S_A^\lambda, \gamma, v)$; by slight abuse of notation we write $E^\lambda = (M^\lambda, \sigma, S_U^\lambda, S_A^\lambda, \gamma, v)$.

A.3 Denial of Service attacks

We assume that the attacker cannot affect the delivery time of user messages. In particular, she cannot perform a denial of service attack by sending messages in a way that causes γ_{ord} to always delay the user's messages. This is implemented by using two separate ordering functions $\gamma_{ord}^0, \gamma_{ord}^1$ for the player with identifier 0 and the player with identifier 1 respectively.

A.4 Mechanism Success

In an asynchronous environment we cannot guarantee that the mechanism correctly identifies the user in any scenario within any bounded time due to the simple fact that the scheduler can delay messages indefinitely. However, to maintain cryptographic security, execution time must be bounded—in a longer execution the

Algorithm 4: Execution main loop of M^λ

```

Execute( $M^\lambda, S_0, C_0, S_1, C_1, \gamma_{ord}, v_\gamma, v_y, v, t$ )
1  $decide_{M^\lambda} \leftarrow \perp$  // Initialize  $M^\lambda$ 's decision variable
2 for  $e \in \{0, 1, M^\lambda\}$  do
3    $Pending_e \leftarrow \emptyset$  // Initialize the message sets
4 for  $t = 0, 1, 2, \dots, \min(t, T(\lambda))$  do
5   for  $e \in \{0, 1, M^\lambda\}$  do // The scheduler chooses which messages to deliver
6      $l_e \leftarrow \gamma_{ord}(e) + [\perp]$ 
7      $Pending_e \leftarrow Pending_e \setminus l_e$ 
8    $Pending'_{M^\lambda} \leftarrow \emptyset$ 
9   for  $msg$  in  $I_0$  do
10     $Pending'_{M^\lambda} \leftarrow Pending'_{M^\lambda} \cup S_0(msg)$  // Player 0 sends messages to the
        mechanism
11   for  $msg$  in  $I_1$  do
12     $Pending'_{M^\lambda} \leftarrow Pending'_{M^\lambda} \cup S_1(msg)$  // Player 1 sends messages to the
        mechanism
13   for  $msg$  in  $I_M$  do
14     $(Pending'_0, Pending'_1) \leftarrow step_{M^\lambda}(msg)$  // The mechanism sends messages to
        the players
15     $Pending_0 \leftarrow Pending_0 \cup Pending'_0$  // Player 0's messages are added to the
        message set
16     $Pending_1 \leftarrow Pending_1 \cup Pending'_1$  // Player 1's messages are added to the
        message set
17    if  $decide_{M^\lambda} \neq \perp$  then // Check if the mechanism reached a decision
18      Return  $decide_{M^\lambda}$  // The chosen player wins
19     $Pending_{M^\lambda} \leftarrow Pending_{M^\lambda} \cup Pending'_{M^\lambda}$  // The mechanism's messages are added
        to the message set

```

Algorithm 5: Execution of M^λ in scenario σ

```

1  $M^\lambda, S_0, C_0, S_1, C_1, v \leftarrow Setup(M^\lambda, \sigma, S_U^\lambda, S_A^\lambda, \gamma, v)$ 
2 Execute( $M^\lambda, S_0, C_0, S_1, C_1, \gamma_{ord}, v_\gamma, v$ )

```

attacker's probability of guessing credentials is not negligible. Previous work (e.g., [43, 44, 65]) sidesteps this issue with protocols that use an ideal signature scheme that cannot be broken indefinitely. We use this abstraction in the rest of the paper. However, in our case this abstraction is not fully satisfactory, as we make claims on general mechanisms using standard communication channels, so for any bounded message length we practically cannot rule out forgery.

Conceptually, we define success as follows. If the mechanism times out in a given execution, we would like to extend the execution and allow it to succeed in the extended execution. To extend the execution without violating cryptographic security, we must increase the security parameter, effectively using a different mechanism. Intuitively, we are interested in *mechanism families* that behave similarly for all security parameters, e.g., verify a signature and send a message independent of the signature details. The requirement is that given such a scheduler, for all sufficiently large security parameters, mechanisms in the family with sufficiently large parameter succeed.

Rather than mechanisms, success is therefore defined for mechanism families that behaves similarly for different security parameters. A *mechanism family* M is a function that maps a security parameter λ to a mechanism M^λ . Similarly, a *strategy family* S is a function that maps a security parameter λ to a strategy S^λ . A mechanism family is *successful* in a scenario σ if, for a large enough security parameter, the user wins against all attacker strategies and schedulers. Formally,

DEFINITION 17 (MECHANISM SUCCESS). A mechanism family M is successful in a scenario σ , if there exists a user strategy family S_U such that for all attacker strategy families S_A , schedulers γ , and random tapes v , there exists a security parameter λ_0 such that:

- (1) for all security parameters $\lambda \geq \lambda_0$, the user wins the execution $E^\lambda = (M^\lambda, \sigma, S_U^\lambda, S_A^\lambda, \gamma, v)$. And,
- (2) for all security parameters λ , either the user wins or the execution times out.

Such a user strategy family S_U is a winning user strategy family in σ with M . Otherwise, the mechanism family fails.

B ONE-SHOT MECHANISMS DOMINANCE - FULL PROOF

Given any mechanism family we construct a dominating one-shot mechanism family by simulating the original mechanism's execution.

CONSTRUCTION 3. For all $\lambda \in \mathbb{N}^+$, we define M_{OS}^λ by specifying the functions $gen_{M_{OS}^\lambda}(\cdot)$ and $step_{M_{OS}^\lambda}(\cdot)$. The credentials' generation function $gen_{M_{OS}^\lambda}(\cdot)$ is the same as $gen_{M^\lambda}(\cdot)$. The mechanism's step function is described in Algorithm 1.

A behavior of M_{OS}^λ is as follows: If it does not receive a message during its execution, it times out. If it receives multiple messages from a player, it ignores all but the first one (lines 2-7). This is implemented using the variables $processing_0$ and $processing_1$ that indicate whether the mechanism has received a message from player 0 and 1, respectively, both initialized to 0. If M_{OS}^λ receives a message that is not a valid strategy and credentials pair, then it decides the identifier of the other player (line 11).

Consider the first message it receives. If it is a valid strategy and credentials pair, then M_{OS}^λ simulates an execution of M^λ (line 14) with the given strategy and credentials while setting both the opponent's strategy and credentials to \perp each (lines 8-9). It uses a scheduler random tape v_γ and an execution random tape v' drawn from v , and an ordering function $\gamma_{ord}^{v_\gamma}$ that chooses the time and ordering of message delivery randomly based on v_γ . If M^λ 's execution decides, then M_{OS}^λ decides the same value. Otherwise, M_{OS}^λ waits for the next message.

If a message arrives from the other player, then similar to the previous case M_{OS}^λ simulates an execution of M^λ with the given strategy and credentials while setting the opponent's to \perp . And again, if M^λ 's simulated execution decides, then M_{OS}^λ decides the same value. Otherwise, M_{OS}^λ waits until $T(\lambda)$ execution steps pass and then times out. So it can decide after receiving the first message from either player or time out.

The mechanism family $M_{OS}(M)$ is one-shot as each of its mechanisms decides based only on the first message it receives from each player. To prove it dominates M , we first show that the attacker's message does not lead to her winning.

LEMMA 7. Let σ be a scenario and let M be a mechanism family successful in σ . Then, for all λ and all executions of $M_{OS}^\lambda(M^\lambda)$ in scenario σ in which the function $step_{M_{OS}^\lambda(M^\lambda)}(\cdot)$ receives a message from the attacker for the first time, either the function sets $decide_{M_{OS}^\lambda}$ to the user's identifier or the simulated execution of M^λ times out.

PROOF. Let σ be a scenario and let M be a mechanism family successful in σ . Let $\lambda \in \mathbb{N}^+$ and consider an execution of M_{OS}^λ in scenario σ in which the function $step_{M_{OS}^\lambda}(\cdot)$ receives an attacker's message for the first time. Denote the identifier of the attacker by $i \in \{0, 1\}$. If the attacker's message is not a valid encoding of a strategy and credentials set, then M_{OS}^λ decides the identifier of the user, and we are done because it sets $decide_{M_{OS}^\lambda}$ to the user's identifier.

Otherwise, the attacker's message is an encoding of a valid strategy and credentials pair $(S_A^\lambda, C_A^\lambda)$. In this case, M_{OS}^λ sets the strategies and credentials to $S_i = S_A^\lambda$, $C_i = C_A^\lambda$, $S_{1-i} = \perp$, and $C_{1-i} = \perp$, $\gamma_{ord}^{v_Y}$, v_Y and v' as in the definition of M_{OS}^λ , and simulates M^λ 's execution by running $Execute(M^\lambda, S_0, C_0, S_1, C_1, \gamma_{ord}^{v_Y} v_Y, v')$. If the simulation times-out or returns the identifier of the user we are done. The only remaining option is that the simulation returns the identifier of the attacker. To show this is impossible, assume by contradiction that the attacker wins in this simulated execution.

First, we show that there exists an execution of M^λ that is identical to the simulated one. Let $\gamma = (\gamma_{ID}^{OS}, \gamma_{ord}^{v_Y} v_Y)$ be a scheduler such that the user gets the same identifier as in the execution of M_{OS}^λ , with the same ordering function and scheduler random tape that M_{OS}^λ uses to simulate M^λ 's main loop. And let v_M be a random tape such that when the execution $E^\lambda = (M^\lambda, \sigma, \perp, S_A^\lambda, \gamma, v_M)$ reaches the main loop, all the next bits of v_M are equal to v' . Note that the main loop of E^λ is the same as the one M_{OS}^λ simulates in E_{OS}^λ . And the attacker wins in the execution E^λ (by the contradiction assumption).

We thus established a simulated execution in which the attacker wins, and in this simulation there exists a time $\tau < T(\lambda)$ when the simulated M^λ decides the identifier of the attacker. Since M is successful in scenario σ , there exists a winning user strategy family S_U . Let γ' be a scheduler such that in the execution $E'^\lambda = (M^\lambda, \sigma, S_U^\lambda, S_A^\lambda, \gamma', v_M)$ it behaves like γ except M^λ receives the user's messages not before τ and before $T(\lambda)$. Such a scheduler exists since the communication is asynchronous and message delivery time is unbounded.

At any time step before τ , the mechanism M^λ sees the same execution prefix whether it is in the execution E^λ with an empty user strategy or in the execution E'^λ with a winning user strategy. Thus, it cannot distinguish between the case where it is in E^λ or E'^λ at τ . Since in E^λ the mechanism M^λ decides at τ , M^λ must decide the same value at τ in E'^λ . That is, the attacker wins also in the execution E'^λ , contradicting the fact that S_U is a winning user strategy for M in σ . Thus, the attacker cannot win in the simulated execution of M^λ . \square

Now we can prove domination.

PROPOSITION 2. *For all profiles π , an authentication mechanism family that solves the π asynchronous authentication problem is dominated by a one-shot mechanism family.*

PROOF. Assume that M is successful in a scenario σ . Then, there exists a user strategy family S_U such that for every attacker strategy family S_A , scheduler γ , and random tape \tilde{v} , there exists a security

parameter λ_{suc} , such that for all $\lambda > \lambda_{suc}$, the user wins the corresponding execution $(M^\lambda, \sigma, S_U^\lambda, S_A^\lambda, \gamma, \tilde{v})$. And for all $\lambda > 0$, either the user wins or the execution times out.

We now show that M_{OS} is successful in scenario σ as well. Denote by S_U^{OS} the user strategy family such that for all $\lambda > 0$, $S_U^{OS, \lambda}$ sends an encoding of the strategy S_U^λ from the winning user strategy family S_U and a set of credentials $C_U \subseteq C_U^\sigma$ that S_U^λ uses in a single message on the first step. And consider any execution of M_{OS}^λ in scenario σ with the user strategy $S_U^{OS, \lambda}$. Note: In the *Setup* function of an execution, the first player in the tuple is always the user and the second is the attacker. However, in the *Execute* function, the first player is the player with identifier 0, which can be the user or the attacker, and the second is the player with identifier 1.

As for all $\lambda > 0$, $S_U^{OS, \lambda}$ sends a message in the first step, there exists a security parameter λ_0^{OS} such that for all $\lambda^{OS} > \lambda_0^{OS}$, the user's message arrives before $T(\lambda)$. It might be the case that an attacker's message arrives first. If the mechanism does not receive any messages, it times out. This is possible only if $\lambda < \lambda_0^{OS}$, as the user strategy we chose does send a message. Otherwise, M_{OS}^λ eventually receives and processes at least one message during its execution, then we consider two cases separately: a user's message arrives first, or an attacker's message arrives first.

Denote by $i \in \{0, 1\}$ the identifier of the player whose message arrives first. If the user's message msg arrives first with the encoded strategy and credential set $(S_U^\lambda, C_U) = extractStrategy(msg)$, then M_{OS}^λ sets the strategies to $S_i = S_U^\lambda$, $C_i = C_U$, $S_{1-i} = \perp$, and $C_{1-i} = \perp$, the scheduler random tape γ_v , the ordering function $\gamma_{ord}^{v_Y}$ and the random tape v' as described in M_{OS} 's definition, and simulates M^λ 's execution by running $Execute(M^\lambda, S_0, C_0, S_1, C_1, \gamma_{ord}^{v_Y} v_Y, v')$. Denote by $\gamma = (\gamma_{ID}^{OS}, \gamma_{ord}^{v_Y} v_Y)$ the scheduler such that the user gets the same identifier as in the execution of M_{OS}^λ , with the same ordering function and scheduler random tape that M_{OS}^λ uses to simulate M^λ 's main loop. Because S_U is a winning user strategy family in M , there exists λ_{suc} such that for all $\lambda > \lambda_{suc}$, the user wins the execution $E^\lambda = (M^\lambda, \sigma, S_U^\lambda, \perp, \gamma, v')$. If $\lambda > \lambda_{suc}$, the simulation terminates and returns the user's identifier i , so M_{OS}^λ also returns i .

Otherwise, as $\lambda \leq \lambda_{suc}$, and because S_U is a winning user strategy family for M^λ , if the simulation terminates then it must return the identifier of the user. And so does the mechanism M_{OS}^λ . If the simulation does not terminate, then M_{OS}^λ waits for the next message from the other player (the attacker). If no message arrives before $T(\lambda)$, then M_{OS}^λ times out as well (note this is possible only if $\lambda \leq \lambda_{suc}$ because otherwise M_{OS}^λ would have already returned the user's identifier). Otherwise, M_{OS}^λ 's step function receives the attacker's message, and by Lemma 1, either M_{OS}^λ decides the user or the simulated execution of M^λ times out. If the simulation times out, then M_{OS}^λ times out as well. Overall we showed that if $\lambda > \lambda_0^{OS}$, a user's message is received by M_{OS}^λ . And if the user's message arrives first, then if $\lambda > \lambda_{suc}$, the user wins, and for all λ , either the user wins or the execution times out.

Now assume the attacker's message arrives first to M_{OS}^λ . Again by Lemma 1, either M_{OS}^λ decides the user or the simulated execution of M^λ in M_{OS}^λ times out. If M_{OS}^λ decides the user, then we are done. Otherwise, the simulation times out, and M_{OS}^λ waits for the next message from the other player (the user). If no other message arrives before $T(\lambda)$, then M_{OS}^λ times out. This is possible only if $\lambda < \lambda_0^{OS}$ by definition of λ_0^{OS} .

Otherwise, M_{OS}^λ receives the user's message with her encoded strategy and credentials set. Then similar to the previous case, M_{OS}^λ sets the strategies, credential sets, scheduler random tape, ordering function, and random tape as described in Construction 1, and simulates M^λ 's execution by running $Execute(M^\lambda, S_0, C_0, S_1, C_1, \gamma_{ord}^{v_\gamma}, v_\gamma, v')$. By the same argument as before, we get that if $\lambda > \lambda_{suc}$, the simulation terminates and returns the user's identifier, so M_{OS}^λ also returns it. Otherwise, $\lambda \leq \lambda_{suc}$, if the simulation terminates, it must return the identifier of the user. And so does the mechanism M_{OS}^λ . If the simulation does not terminate, then M_{OS}^λ times out.

Overall we showed that if $\lambda > \max(\lambda_{suc}, \lambda_0^{OS})$, the user wins. And for $\lambda \leq \max(\lambda_{suc}, \lambda_0^{OS})$, either the user wins or the execution times out. Therefore, M_{OS} is successful in scenario σ . We thus conclude that the one shot mechanism family M_{OS} dominates M . \square

C DETERMINISTIC MECHANISMS ARE DOMINATED BY BOOLEAN MECHANISMS

Lemma 4 (restated). *Let M_{det} be a deterministic credential-based mechanism and let the function f be as constructed above. Then f is monotonic.*

PROOF. Let M_{det} be a deterministic mechanism and let f be as defined in Construction 2. We prove that if M_{det} is successful in a scenario, then M_f is successful as well. Assume M_{det} is successful in scenario $\sigma = (\sigma_U, \sigma_A)$, by definition of f , we get that $f(\sigma_U) = 1$.

As M_{det}^λ is successful in σ , there exists a winning user strategy family S_U^{det} for M_{det} . And because M_{det} is a credential-based mechanism, for each λ , $S_U^{det, \lambda}$ sends a set of credentials c_U^λ in a single message. Let $\lambda \in \mathbb{N}^+$ be any security parameter, let $S_U^{f, \lambda}$ be the user strategy for M_f^λ that sends the same subset of the user's credentials as $S_U^{det, \lambda}$ in a single message in the first step. Let S_A^f be an attacker strategy family for M_f , γ_f a scheduler, and v a random tape.

Consider the execution $E_f^\lambda = (M_f^\lambda, \sigma, S_U^{f, \lambda}, S_A^f, \gamma_f, v)$. Because $S_U^{f, \lambda}$ sends a message in the first step, there exists a security parameter λ_0^f such that for all $\lambda > \lambda_0^f$, the user's message arrives before $T(\lambda)$. It might be the case that an attacker's message arrives first. If the mechanism does not receive any messages, it times out. This is possible only if $\lambda \leq \lambda_0^f$, by definition of λ_0^f .

Otherwise, M_f^λ eventually receives and processes at least one message during its execution, then we consider two cases separately: a user's message arrives first, or an attacker's message arrives first. If the user's message arrives first to M_f^λ , with her set of credentials, then as $f(\sigma_U) = 1$ (because $\sigma \in \Pi(M_{det})$), M_f^λ decides the user. We

get that if $\lambda > \lambda_0^f$, then M_f^λ decides the user. And if $\lambda \leq \lambda_0^f$, then M_f^λ either decides the user or times out. Overall we showed that if the user's message arrives first, then if $\lambda > \lambda_0^f$ the user wins, otherwise, either the user wins or the execution times out. So $\lambda_{suc}^f = \lambda_0^f$.

Otherwise, the attacker's message arrives first to M_f^λ . If $f(\sigma_A) = 0$, then M_f^λ decides the user, and we are done, because the user wins. The only other option is if $f(\sigma_A) = 1$. We show that this is not possible.

By definition of f , $f(q) = 1$ if and only if there exists a scenario $\sigma_{U:q}$ in which the user's availability vector is q and M_{det} succeeds in $\sigma_{U:q}$. Let $S_{U:q}$ be the winning user strategy for M_{det} in $\sigma_{U:q}$. Then for all attacker strategy families S_A , schedulers $\gamma = (\gamma_{ID}, \gamma_{ord}, v_\gamma)$ and random tapes v , there exists a security parameter λ_{suc} such that for all $\lambda > \lambda_{suc}$, the user wins the execution $E_{U:q}^\lambda = (M_{det}^\lambda, \sigma_{U:q}, S_{U:q}^\lambda, S_A^\lambda, \gamma, v)$. This includes the attacker strategy that does not send any messages $S_A^\lambda = \perp$. Then during the execution $E_{U:q}^\lambda$'s main loop there exists a time $\tau < T(\lambda)$ when M_{det}^λ decides the identifier of the user.

Now consider the execution of M_{det}^λ in σ , in which the attacker uses the same strategy as the users' $S_{U:q}^\lambda$ from $\sigma_{U:q}$. This is possible as the attacker has access to all credentials the user had in $\sigma_{U:q}$. Let γ' be a scheduler in which the user and the attacker's identifiers are opposite to the ones in γ , the attacker's message arrives first and the user's message is delayed beyond τ . And let v' be any random tape. Denote this execution by E^λ .

At any time step before τ , the mechanism M_{det}^λ sees the same execution prefix whether it is in E^λ or in $E_{U:q}^\lambda$, and thus, it cannot distinguish between the case where it is in E^λ or $E_{U:q}^\lambda$. As M_{det}^λ decides the user in $E_{U:q}^\lambda$'s main loop at time τ , then it must decide the attacker at the same time τ in E^λ 's main loop (as the identifiers are reversed, and the mechanism sees the exact same execution prefix). Therefore, M_{det}^λ decides the attacker in E^λ contradicting the assumption that M_{det}^λ succeeds in σ . We conclude that $f(\sigma_A) = 0$ and M_f^λ decides the user also in this case. Therefore, M_f is successful in σ . Overall we showed that the mechanism family M_f is a monotonic Boolean mechanism family and dominates M_{det} . \square

D SUCCESS EQUIVALENCE

LEMMA 8. *Let f be a monotonic Boolean function, and let M_f be the mechanism family of f . M_f is successful in a scenario σ if and only if $f(\sigma_U) = 1$ and $f(\sigma_A) = 0$.*

PROOF. For the first direction, assume that M_f is successful in a scenario σ . Then, there exists a winning user strategy family S_U for M_f in σ that sends her set of credentials in a single message on the first step. For every attacker strategy S_A , scheduler γ and random tape v , there exists a security parameter λ_{suc} such that for all $\lambda > \lambda_{suc}$ the user wins the execution $(M_f^\lambda, \sigma, S_U, S_A, \gamma, v)$.

Because S_U^λ sends a message in the first step, there exists a security parameter λ_0^f such that for all $\lambda > \lambda_0^f$, the user's message eventually arrive before $T(\lambda)$. It might be the case that an attacker's

message arrives first. If the mechanism does not receive any messages, it times out. This is possible only if $\lambda < \lambda_0^f$, as the user strategy we chose always send a message.

Let $\lambda > \lambda_0^f$ and consider a scheduler γ for which the user's message arrives first. Consider the execution $E = (M_f^\lambda, \sigma, S_U, S_A, \gamma, v)$. As the user's message arrives first, with her set of credentials, M_f will extract the availability vector σ_U of the credentials. Then, as M_f is successful in σ , it will return the identifier of the user. This can happen only if $f(\sigma_U) = 1$.

Now let S_U be a winning user strategy and S_A an attacker strategy that sends any subset of the attacker's credentials. Let γ' and v' be a scheduler and random tape such that the attacker's message arrives first, and consider the execution $E' = (M_f^\lambda, \sigma, S_U, S_A, \gamma', v')$. In that case, as the attacker sends a subset of her credentials, M_f^λ will extract the availability vector σ'_A of the credentials. Then, as M_f is successful in σ , it will return the identifier of the user. This is possible only if $f(\sigma'_A) = 0$. This is true for any subset of credentials the attacker chooses, including the set of all her credentials. Thus, $f(\sigma_A) = 0$. Therefore, we showed that if M_f is successful in a scenario σ , then $f(\sigma_U) = 1$ and $f(\sigma_A) = 0$.

For the second direction, assume that $f(\sigma_U) = 1$ and $f(\sigma_A) = 0$. Let S_U be a user strategy family that sends the set of credentials corresponding to σ_U in the first step. And let S_A, γ and v be any attacker strategy, scheduler and random tape. We show that S_U is a winning user strategy for M_f in σ .

Consider the execution $(M_f^\lambda, \sigma, S_U, S_A, \gamma, v)$. If no message was received, then M_f^λ times out. This is possible only if $\lambda < \lambda_0^f$, as the user strategy we chose sends a message on the first step. Otherwise, M_f^λ receives a message.

If the user's message arrives first to M_f^λ with the credentials corresponding to σ_U , then M_f^λ extracts the availability vector σ_U of the credentials. Then, as $f(\sigma_U) = 1$, the mechanism M_f^λ returns the identifier of the user, thus the user wins the execution. Otherwise, the attacker's message msg_A arrives first to M_f^λ . If msg_A does not contain a valid set of credentials, then M_f^λ returns the identifier of the user, and she wins the execution. Otherwise, M_f^λ extracts the availability vector $q \leq \sigma_A$ of the credentials in msg_A (the attacker might send any subset of credentials she knows). Then, because f is monotonic and $f(\sigma_A) = 0$, also $f(q) = 0$ and M_f^λ returns the identifier of the user, thus the user wins the execution. In both cases, if $\lambda > \lambda_0^f$, the execution terminates and the user wins. And if $\lambda < \lambda_0^f$, either the user wins or the execution times out. Thus, M_f is successful in σ , concluding the proof that M_f is successful in a scenario σ if and only if $f(\sigma_U) = 1$ and $f(\sigma_A) = 0$. \square

E PARTIAL BOOLEAN MECHANISMS

Observation 2 (restated). *Let M_1 and M_2 be two mechanisms such that $\emptyset \neq \Pi(M_1) \subseteq \Pi(M_2)$. Let T_1 and T_2 be the sets of all user availability vectors and F_1 and F_2 be the sets of all attacker availability vectors in $\Pi(M_1)$ and $\Pi(M_2)$ respectively. Then, $T_1 \subseteq T_2$ and $F_1 \subseteq F_2$.*

PROOF. Let M_1, M_2, T_1, T_2, F_1 and F_2 be as in the claim. We show that, $T_1 \subseteq T_2$ and $F_1 \subseteq F_2$. First, note that as $\Pi(M_1) \neq \emptyset$, then T_1 and F_1 are both not empty. Because the profile of a mechanism is the Cartesian product of the set of user availability vectors (T_1) with the set of attacker availability vectors (F_1) (Observation 1). Similarly, T_2 and F_2 are both not empty. Assume for contradiction this at least one of the following holds: $T_1 \not\subseteq T_2$ or $F_1 \not\subseteq F_2$.

Without loss of generality, assume that $T_1 \not\subseteq T_2$. As $\Pi(M_1) \neq \emptyset$, there exists $q \in T_1$ such that $q \notin T_2$. Consider a scenario σ such that $\sigma_U = q$ and $\sigma_A \in F_1$. Then, $\sigma \in \Pi(M_1)$ and $\sigma \notin \Pi(M_2)$, contradicting the fact $\Pi(M_1) \subseteq \Pi(M_2)$. Thus, $T_1 \subseteq T_2$. Similarly, $F_1 \subseteq F_2$. \square

Lemma 5 (restated). *For all mechanisms, there exists an equivalent monotonic partial Boolean mechanism.*

PROOF. Let M be a mechanism with n credentials. If M 's profile is empty, then M is equivalent to the Boolean mechanism of constant Boolean function $f(x) = 0$. A constant function is monotonic, and thus we are done.

Otherwise, by Theorem 1, there exists a monotonic Boolean function f such that the Boolean mechanism M_f dominates M . That is, $\Pi(M) \subseteq \Pi(M_f)$. Let T be the set of all user availability vectors in $\Pi(M)$ and F be the set of all attacker availability vectors in $\Pi(M)$.

By definition, To prove that (T, F) is a partial Boolean function, we must show that $T \cap F = \emptyset$. Let T_f and F_f be the sets of all user and attacker availability vectors in $\Pi(M_f)$ respectively. As M and M_f 's profiles are not empty, by Observation 2, we get that $T \subseteq T_f$ and $F \subseteq F_f$.

We show that $T \cap F = \emptyset$. If there exists a vector $q \in T \cap F \subseteq T_f \cap F_f$, then for q , it holds that $f(q) = 1$ and $f(q) = 0$, contradicting the fact that f is a well-defined function. Thus, $T \cap F = \emptyset$ and (T, F) is a partial Boolean function. It is monotonic because f is an extension of (T, F) and f is monotonic. And the mechanism M is equivalent to the monotonic partial Boolean mechanism of (T, F) . \square

F PROFILE SIZE BOUNDS

We calculate the number of viable scenarios.

Observation 3 (restated). *The number of viable scenarios for n credentials is $4^n - 3^n$.*

PROOF. The number of scenarios for n credentials is 4^n . The number of viable scenarios is the number of scenarios in which there exists at least one safe credential. The number of scenarios in which there are no safe credentials is the number of scenarios in which all credentials are either lost, leaked, or stolen. That is, 3^n scenarios. Therefore, the number of viable scenarios is $4^n - 3^n$. \square

We prove an upper bound on the number of scenarios that can be added to a profile of a partial Boolean function.

Observation 4 (restated). *Let (T, F) be a partial Boolean function of n credentials. Let $s = \max(|T|, |F|)$. The maximum number of scenarios that can be added to the profile of the mechanism $M_{(T,F)}$ without contradicting it is $s \cdot (2^n - s) - |\Pi(M_{(T,F)})|$ if $s \geq 2^{n-1}$ and $4^n - 3^n - |\Pi(M_{(T,F)})|$ otherwise.*

PROOF. Let (T, F) be a partial Boolean function of n credentials and let $s = \max(|T|, |F|)$. For n credentials, the number of Boolean vectors is 2^n . By Observation 1, we have $|\Pi(M_{(T,F)})| = |T| \cdot |F|$. There are three separate cases:

(1) If $s \geq 2^{n-1}$ and $s = |T|$ then the maximal size of a profile is reachable if all vectors $v \notin T \cup F$ are added to F . That is, the number of scenarios that can be added to the profile is at most $|T| \cdot (2^n - |T|) - |\Pi(M_{(T,F)})|$

(2) If $s \geq 2^{n-1}$ and $s = |F|$ the case is symmetric to the previous one and the number of scenarios that can be added to the profile is at most $|F| \cdot (2^n - |F|) - |\Pi(M_{(T,F)})|$

(3) Otherwise, we get that $s < 2^{n-1}$, then the maximal size of a profile is $4^n - 3^n$, the same as the number of viable scenarios (Observation 3) as by [35], non-viable scenarios are not in the profile. In this case, the number of scenarios that can be added to the profile is at most $4^n - 3^n - |\Pi(M_{(T,F)})|$. \square

G ALGORITHM ANALYSIS

G.1 Algorithm Correctness

We first prove that our algorithm keeps the (possibly partial) truth table monotonic after each update.

CLAIM 1. *After each update, the partial monotonic truth table remains a (possibly partial) monotonic truth table, and the above update does not contradict previous ones.*

PROOF. Given a monotonic partial truth table, it holds that for all $x, y \in \{0, 1\}^n$ such that $x > y$ we have that if $f(y) = 1$ then $f(x) = 1$, and if $f(x) = 0$ then $f(y) = 0$. According to the algorithm, in every step where we add a viable scenario σ , we check that σ_U is not set to 0 and σ_A is not set to 1. If so, we set σ_U to 1 and σ_A to 0. As σ is viable, we have that $\sigma_U \not\leq \sigma_A$, so this update preserves monotonicity.

Because σ_U was not set to 0 before, for all $x > \sigma_U$ we have that $f(x) \neq 0$, and we can set them to 1. Similarly, because σ_A was not set to 1 before, for all $y < \sigma_A$, $f(y) \neq 1$, and we can set them to 0. Therefore, the truth table can still be completed into a monotonic Boolean function. \square

We now prove our algorithm correctness, including termination, the truth table validity, and bound the distance of the returned mechanism from the optimal one.

LEMMA 9. *Let n be the number of credentials and let $0 \leq \delta < 1$. Then when Algorithm 2 returns, the variable $maxTable$ contains a truth table of a mechanism that is at least δ close to the optimal mechanism.*

PROOF. We divide the proof into three parts. First, we show that the algorithm always stops. Second, we show that when the algorithm returns, $maxTable$ contains a valid complete monotonic truth table. Finally, we show that the mechanism defined by the truth table is at least δ close to the optimal mechanism.

To show that the algorithm always stops, we note that the number of scenarios that can be added to a profile of a partial Boolean mechanism is bounded. Thus, in the worst case, the algorithm explores all possible scenarios, which is a finite number. Once no

scenarios are left, the condition in line 11 is met, and the algorithm stops.

To show that the algorithm always results in a valid complete monotonic truth table, we note that by Claim 1, the truth table remains a (possibly partial) monotonic truth table after each update. Thus, it is sufficient to show that the algorithm updates $maxTable$ to a valid complete monotonic truth table at least once, and never updates it to a non-complete truth table.

The algorithm stops the recursive exploration only when one of the three stopping conditions is met. In the first case (line 3), the truth table is complete, and the algorithm updates $maxTable$ to this complete table. In the second case (line 11), the algorithm completes the truth table arbitrarily, and $maxTable$ gets the completed table. Finally, in the third case (line 18), the algorithm prunes the branch, and returns, without updating $maxTable$. Thus, if the algorithm updates $maxTable$, it always updates it to a complete valid truth table.

To show that the algorithm updates $maxTable$ at least once, we note that the third stopping condition (line 18) is the only one that does not update $maxTable$. However, this condition is met only if $maxSuccessProb$ is greater than 0. And as $maxSuccessProb$ is initialized to 0, and $maxTable$ and $maxSuccessProb$ are updated only together (lines 5-6 and 14-15), the algorithm must return at least once because one of the first two stopping conditions is met. Thus, $maxTable$ is updated to a valid complete monotonic truth table.

To complete the proof, we show that the mechanism defined by the truth table's success probability is at most δ lower than the optimal mechanism. Denote by M the mechanism defined by the truth table $maxTable$ after the algorithm returns, with success probability $maxSuccessProb$, and by M^* the optimal mechanism. Assume for contradiction that the probability $maxSuccessProb + \delta$ is smaller than the success probability of M^* . By Theorem 1, the optimal mechanism M^* can be represented by a (complete) monotonic Boolean function. Therefore, there exists a path along the recursive exploration tree that leads to the optimal mechanism, which the algorithm pruned (in line 19).

Consider the function call in which the algorithm pruned the path to the optimal mechanism. From Observation 4, we know that $numPossibleAdditions$ (line 9) is an upper bound on the number of scenarios that can be added to the current truth table without contradicting it. As $potentialSuccessProb$ is calculated by summing the probabilities of the current partial truth table and the next $numPossibleAdditions$ scenarios with the highest probabilities, then in this branch, $potentialSuccessProb \geq M^*$'s success probability. However, as the algorithm pruned this branch, we know that $maxSuccessProb > potentialSuccessProb - \delta$ (line 18). Therefore, $maxSuccessProb + \delta > M^*$'s success probability. Contradicting the assumption that $maxSuccessProb + \delta$ is smaller than the success probability of M^* . Therefore, the algorithm returns a mechanism that is at most δ away from the optimal mechanism.

Overall, we showed that the algorithm returns a valid complete monotonic truth table representing a mechanism whose success probability is at most δ lower than the optimal mechanism. \square

G.2 Algorithm Complexity

We first discuss the difficulty of calculating the exact complexity of the algorithm in a general case (§5.5.1). Then we evaluate the complexity of the algorithm empirically (§5.5.2), by measuring the runtime of our algorithm for different numbers of credentials and fault probabilities.

G.2.1 Bound. To give a loose upper bound on a single function call of the algorithm’s complexity, we analyze the complexity of each step separately. First, to calculate the current profile (line 1), we use a Cartesian product of the availability vectors of the user and the attacker as described in Observation 1. By [35] and Observation 3, this is bounded by $O(4^n - 3^n)$. Calculating the success probability of the current truth table (line 2) requires summing the probabilities of the scenarios in the profile ($O(4^n - 3^n)$). Finding and sorting the possible additional scenarios probabilities (line 8) is $O((4^n - 3^n) \cdot n)$. Finding the number of possible additions (line 9) is calculated based on Observation 4 and requires $O(1)$. The bound itself is $O(4^n - 3^n)$ in the worst case, if we could add all of the viable scenarios. Therefore, calculating the sum of the probabilities of the next $numPossibleAdditions$ scenarios (line 10) is $O(4^n - 3^n)$. Excluding the function calls, all other steps are $O(1)$. Thus, the complexity of a single recursive function call is $O((4^n - 3^n) \cdot n)$. And in the worst case, the recursion depth may reach $O(2^{4^n - 3^n})$. Resulting in a total complexity of $O(2^{4^n - 3^n} \cdot (4^n - 3^n) \cdot n)$.

But this analysis assumes the worst case for every step of the algorithm. However, in practice, many steps’ worst case cannot happen simultaneously. Thus, the actual complexity of the algorithm is much lower than the calculated upper bound.

Calculating the exact complexity of the algorithm depends on the number of credentials, their specific probabilities, and the parameter δ . While an upper bound on the recursion depth is given by 2 to the power of the number of viable scenarios, the actual depth explored is much smaller. This is due to multiple factors, for example (1) the exponential drop in the probability of different scenarios, combined with the fact that the algorithm prunes branches with negligible advantage; (2) the fact that the number of scenarios that can be added to a profile of a partial Boolean mechanism is limited (Observation 4) in a non-trivial way depending on which do result in a monotonic mechanism; and (3) each scenario adds not a single row to the truth table, but possibly many rows for keeping monotonicity (depending on the specific scenario and the current truth table). While all those factors contribute to the algorithm’s efficiency, it remains an open question whether the algorithm’s exact complexity can be calculated theoretically (cf. the lack of a closed form expression for the number of monotonic Boolean functions [15]).

G.2.2 Empirical Complexity. We provide additional examples of our algorithm runtime behavior in Table 1 and Figure 6.

Now we describe each plot in Figure 6 and the corresponding fitted functions in Table 1, categorized by the runtime function behavior.

Exponential Growth: When one credential can suffer from up to a single type of fault and the rest can have up to two types of faults, or when all credentials can suffer from all types of faults with low probability, the algorithm’s runtime grows exponentially

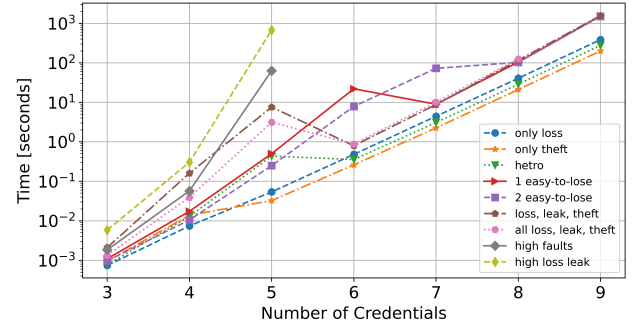


Figure 6: Runtime of the algorithm as a function of the number of credentials for different fault probabilities with $\delta = 10^{-5}$.

($O(4^n)$) with the number of credentials (all fits with $R^2 > 0.97$, exact values are in Table 1).

For example, when all credentials can suffer from loss with $p^{loss} = 0.01$, $p^{leak} = p^{theft} = 0$ (*only loss* in Figure 6), or all credentials can suffer from theft with $p^{theft} = 0.01$, $p^{loss} = p^{leak} = 0$ (*only theft* in Figure 6).

When one key is prone to loss and leak with $p^{loss} = p^{leak} = 0.01$, $p^{theft} = 0$ and the rest can only be stolen with $p^{theft} = 0.01$ (*hetro* in Figure 6). Or when one credential can be easily lost, but not leaked or stolen, with $p_1^{loss} = 0.3$, $p_1^{leak} = p_1^{theft} = 0$ and the rest can be either lost or leaked with $j > 1$, $p_j^{loss} = p_j^{leak} = 0.01$, $p_j^{theft} = 0$. (*1 easily-to-lose* in Figure 6). Similarly, when 2 credentials can be easily lost, but not leaked or stolen (*2 easily-to-lose* in Figure 6).

We observe a similar trend when each credential can have all three types of faults, where at least two with low fault probabilities, $p^{loss} = 0.01$, $p^{leak} = p^{theft} = 0.001$ (*loss, leak, theft* in Figure 6), or when one credential can be only lost with $p_1^{loss} = 0.01$, $p_1^{leak} = p_1^{theft} = 0$ and the rest can have all three types of faults with $i > 1$, $p_i^{loss} = 0.01$, $p_i^{leak} = p_i^{theft} = 0.001$ (*all loss, leak, theft* in Figure 6).

Super-Exponential Growth: When all credentials can suffer from two or more types of faults with a high probability, the algorithm’s runtime grows super-exponentially with the number of credentials. E.g., if all three fault types are possible with high probabilities, say $p^{loss} = 0.1$, $p^{leak} = 0.3$, $p^{theft} = 0.4$ (*high faults* in Figure 6). Or when all credentials are pruned to two or more types of faults with high probabilities (e.g., $p^{loss} = p^{leak} = 0.01$, $p^{theft} = 0$) (*high loss leak* in Figure 6), the algorithm’s runtime grows too rapidly to obtain sufficient data points for regression analysis.

Table 1: Runtime Analysis

Name	Loss	Leak	Theft	Formula	R^2
only loss	10^{-2}	0	0	$0.0014 \cdot 4^n - 12.01$	0.979
only theft	0	0	10^{-2}	$0.0007 \cdot 4^n - 6.20$	0.979
hetro	$P_1 = 10^{-2}, i > 1 : P_i = 0$	$P_1 = 10^{-2}, i > 1, P_i = 0$	$P_1 = 0, i > 1, P_i = 10^{-2}$	$0.001 \cdot 4^n - 8.68$	0.979
1easy-to-lose	$P_1, i > 1 : P_i = 10^{-2}$	$P_1 = 0, i > 1, P_i = 10^{-2}$	0	$0.0059 \cdot 4^n + -53.54$	0.967
2easy-to-lose	$P_1 = P_2 = 0.3, i > 2 : P_i = 10^{-2}$	$P_1 = P_2 = 0, i > 2, P_i = 10^{-2}$	0	$0.0056 \cdot 4^n - 42.99$	0.968
loss, all-leak	$P_1 = 0, i > 1 : P_i = 10^{-2}$	10^{-2}	0	$0.0004 \cdot 4^n - 0.41$	0.977
loss, leak, theft	10^{-2}	10^{-3}	10^{-3}	$0.006 \cdot 4^n - 55.44$	0.97
all loss, leak, theft	10^{-2}	$P_1 = 0, i > 1, P_i = 10^{-3}$	$P_1 = 0, i > 1, P_i = 10^{-3}$	$0.0019 \cdot 4^n - 4.69$	0.97