Drifting Keys: Impersonation detection for constrained devices

Drifting Keys: Impersonation Detection for Constrained Devices

Kevin D. Bowers and Ari Juels
RSA Laboratories
Cambridge, MA, USA
Email: {kbowers, ajuels}@rsa.com

Ronald L. Rivest and Emily Shen
MIT CSAIL
Cambridge, MA, USA
Email: {rivest, ehshen}@mit.edu

***Abstract*—We introduce *Drifting Keys* (DKs), a simple new approach to detecting device impersonation. DKs enable detection of *complete* compromise by an attacker of the device and its secret state, e.g., cryptographic keys. A DK evolves within a device randomly over time. Thus a clone device created by the attacker will emit DKs that randomly diverge from those in the original, valid device over time, alerting a trusted verifier to the attack.**

DKs may be transmitted unidirectionally from a device, eliminating interaction between the device and verifier. Device emissions of DK values can be quite compact—even just a single bit—and DK evolution and emission require minimal computation. Thus DKs are well suited for highly constrained devices, such as sensors and hardware authentication tokens.

We offer a formal adversarial model for DKs, and present a simple scheme that we prove essentially optimal (undominated) for a natural class of attack timelines. We explore application of this scheme to one-time passcode authentication tokens. Using the logs of a large enterprise, we experimentally study the effectiveness of DKs in detecting the cloning of such tokens.

I. Introduction

A central tenet of modern cryptography is that well designed primitives should rely not on algorithmic secrecy, but on the confidentiality of entities' keys. Thus, cryptography today hinges on what are called *secret* (symmetric) or *private* (asymmetric) keys. Overwhelmingly, cryptographic primitives are designed to respect these labels on the assumption that keys will *remain* secret or private.

Many threats, though, are undermining the inviolacy of cryptographic keys. Side-channel attacks bypass logical-layer protections and can extract device keys via techniques such as timing analysis [12] and differential power analysis [13]—even wirelessly [18]. Hardware tampering is a longstanding challenge [2], as is widespread, poor implementation of cryptography [16]. Worse still, recent high-profile breaches [20], [21] illustrate the threat of *en bloc* compromise of sensitive data, the theft of entire databases of keys or other secrets.

A good illustration of this threat and motivation for our work is the 2011 breach of security company RSA affecting its well known SecurID authentication tokens [23]. While RSA declined to reveal specifics, its replacement of millions of tokens fueled speculation that information was compromised relating to token "seeds," i.e., their cryptographic keys [9]. The breach purportedly later led to an attempted token-cloning attack against defense contractor Lockheed-Martin [24].

In this paper, we consider the following problem. A small *device*, e.g., a hardware authentication token belonging to a user, contains a cryptographic key K. The device uses K to authenticate messages (or itself) in transmissions to a trusted *verifier*, e.g., an authentication server. An attacker breaches the device—or the server it interacts with—and compromises, i.e., steals key K. The attacker then *impersonates* the device, e.g., simulates the original token to authenticate as the honest user. Because the attacker knows K and any other device state, it can impersonate the device perfectly: Its transmissions are identical to the original device's, and the attack is undetectable. Our goal in this paper is to detect such attacks, which we interchangeably call impersonation or *cloning*.

Key rotation is a basic, common defense against key compromise. A device's cryptographic key K_t might be valid only for some time period t, and replaced (typically by the verifier) with a fresh key K_{t+1} at time $t+1$. If an attacker steals K_t at time t and tries to impersonate the device in time period $t+1$, it is likely to fail (and its attempt is detectable).

Ordinary key rotation, however, is hard to implement in constrained devices. RSA SecurID tokens, for instance, are highly bandwidth constrained. They display a passcode (typically eight decimal digits) that a user transcribes to authenticate to a remote service, and they take *no* input.

Our contribution is a new, conceptually simple impersonation detection scheme called *Drifting Keys*. (We abbreviate as DK.) A DK κ_t is a (short) extra key stored in a device that is independent of any ordinary cryptographic device key K and its functionality. The device itself rotates κ_t, i.e., *randomly* changes it at time $t+1$ to obtain a new DK κ_{t+1}. The device uses a true random source to rotate its keys; thus, an adversary that has stolen κ_t is unlikely to be able to generate the same κ_{t+1} as the device. The device communicates its changes in DK to the verifier; the verifier detects impersonation attacks when it receives inconsistent DKs.

DKs differ from ordinary key rotation in their need to meet three distinct challenges:

1) *Gradual rotation:* Small devices often have little bandwidth, and can't easily send or receive full, fresh keys. DKs must thus be *gradually* rotated, undergoing randomization of just one or a few DK bits per time period.
2) *Unidirectionality:* As many small devices are output-only (e.g., SecurID tokens), DKs must require only transmis-

sion from a device to a verifier, not the reverse.

3) *Loose synchronization:* Message loss is common in small-device settings. DKs must thus be designed to be robust to a lossy channel between the device and verifier.

DKs can detect cloning in many different scenarios. At the level of individual devices, for example, if a node in a sensor network is captured, compromised, and cloned, DKs can detect resulting bogus nodes in the network. DKs are also useful for systemic attacks. In an enterprise authentication-token system, for instance, a server-side breach can divulge keys for many (or all) tokens. DKs can help detect attacks on individual tokens that cumulatively yield systemwide insight into the breach.

DKs are conceptually simple, but three challenges above make their implementation tricky. Also, given the constrained bandwidth and lossy channels of small device settings, verifiers must be able to detect cloning with only *partial* DK information. And as we show in our study of DKs in authentication tokens in particular, embedding DKs into legacy systems poses challenges of practical interest.

Our contributions in this paper are thus as follows:

- **Introducing DKs:** We introduce and formally model Drifting Key schemes.
- **Optimal DK scheme:** We prove that a simple DK scheme ("Uniformly Staggered") is the best possible (formally, is undominated) for a natural class of timelines.
- **DK Compression:** We present a coding scheme (what we call "views") that enables the use of long DKs on a device-to-verifier channel that is bandwidth-constrained, lossy, and subject to (limited) adversarial eavesdropping.
- **Application to authentication tokens:** We explore the application of drifting keys to commercial one-time passcode authentication tokens. We show how to overcome legacy and usability challenges and demonstrate practicality via simulation on real-world enterprise data.

Organization Section II presents an overview of DK schemes and challenges and our roadmap. Section III specifies our formal model, while Section IV gives our main DK scheme ("Uniformly Staggered"). We show how to compress DKs to meet bandwidth constraints in Section V. In Section VI we explore practical application of DKs to authentication tokens, with experiments on real-world enterprise data. We review related work in Section VII and conclude in Section VIII.

II. Overview and Roadmap

In general, DKs help detect a cloning attack in a scenario such as the following. An attacker transiently compromises the device (or verifier) at some time A, learning *all* of the device's secret keys, including its current DK κ_A. The device continues to evolve its DK randomly and to transmit them to the verifier (without the presence of the attacker). Its last such transmission, of DK κ_B, happens at time $B \geq A$. At some later time $C \geq B$, the attacker attempts to impersonate the device by transmitting a valid DK $\tilde{\kappa}_C$ to the verifier.

The hope is that the verifier will identify $\tilde{\kappa}_C$ as a forgery. But the verifier doesn't actually know the correct, current device DK κ_C. In order to check an incoming DK at time C, the verifier must *extrapolate* from its last sighted valid DK κ_B. In particular, the verifier may determine a *range* R of possible, valid values for the device's current DK κ_C. If the DK scheme is successful, the attacker's forged DK $\tilde{\kappa}_C$ will be inconsistent with κ_B, i.e., $\tilde{\kappa}_C \notin \mathsf{R}$. The verifier will then detect the attacker's cloning attempt. Figure 1 schematically depicts this sequence of events in a DK system.

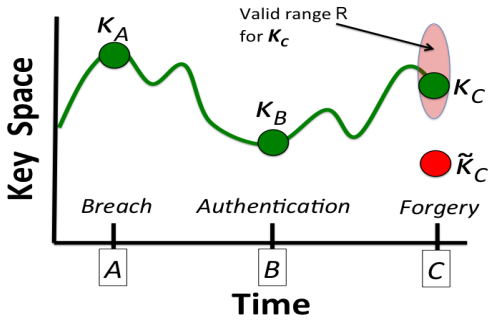

Fig. 1. Intuition behind DKs. At time A, an attacker breaches the system, learning DK κ_A. At time B, the device sends current DK κ_B to the verifier. At time C, the attacker presents forged DK $\tilde{\kappa}_C$. The verifier expects to receive a key κ_C from range R (the range of valid drift of κ_B over interval $[B, C]$.)

Example DK scenario: Here is an illustration of DK use:

Example 1: A motion sensor sends readings to a base station at the beginning of each hour, digitally signed (MACed) using secret key σ.

To facilitate clone detection, a DK is appended to sensor outputs. Let κ_t be the DK appended in hour t. This DK grows by one random bit per hour. For hour $t+1$, a fresh, random bit $\kappa[t+1]$ is appended to κ_t to yield κ_{t+1}.

An attacker (transiently) breaches the sensor in hour $t = 2$, learning both the key σ and $\kappa_2 = \kappa[1]\kappa[2]$. The attacker then tries to forge a sensor output at the end of hour $t = 3$, after the sensor has sent $\kappa_3 = \kappa[1]\kappa[2]\kappa[3]$. I.e., the attacker attempts to pass off a forged DK $\tilde{\kappa}_3 = \kappa[1]\kappa[2]\tilde{\kappa}[3]$ as valid.

To succeed and submit a correct DK $\tilde{\kappa}_3 = \kappa_3$, the attacker must correctly guess $\tilde{\kappa}[3]$, i.e., set $\tilde{\kappa}[3] = \kappa[3]$. It will fail, and the base station will reject $\tilde{\kappa}_3$, with probability at least $1/2$.

(In this simple example, $\mathsf{R} = \{\kappa_3\}$, i.e., the range of valid DKs at time C is just one key.)

Example 1 considers a simple DK design and attack scenario. DKs grow one bit at a time, indefinitely, and are transmitted in their entirety. And $C = B$, so the device (sensor) has synchronized its DK with the verifier (base station) at the time of forgery C (i.e., $C = B$). In practical settings, it often isn't feasible to transmit full DKs (or maintain arbitrarily growing ones). And the device and verifier are often not synchronized (i.e., $C > B$), so that the verifier must detect attacks with *partial* information—or even after the fact. These two challenges are major ones addressed in this paper.

Roadmap: We analyze the security of DK schemes in this paper in terms of a *timeline* characterizing an attack—in particular, the interval lengths $[A, B]$ and $[B, C]$. In Section III,

we introduce a formal model, expressed as a standard cryptographic adversarial experiment. This model enables us to determine the success probability of an attacker for a given DK scheme on a given timeline (or class of timelines).

As in Example 1 above, one objective of DK schemes is *prevention*, i.e., authentication rejection when an attacker tries to submit an invalid DK. The timeline in Figure 1 shows a prevention scenario.

When it receives two incompatible DKs, however, the verifier doesn't know *a priori* which is correct, and which is invalid. The verifier knows only that it has detected a cloning attempt. Thus another sequence of interest is one in which an attacker first forges a DK $\tilde{\kappa}_B$ and the device later makes an authentication attempt, submitting DK κ_C. If the verifier then determines that $\tilde{\kappa_B}$ is inconsistent with κ_C, then it's too late to prevent the attacker from authenticating, but the verifier achieves *detection* of the attack—also very valuable. Our model in Section III covers both prevention and detection, and we study DK schemes with respect to both.

In Section IV, we introduce a particular DK scheme that is our main focus in this paper, and which we call *Uniformly Staggered* (US). This is a simple scheme in which a DK consists of m keys, i.e., $\kappa_t = \kappa[1], \ldots, \kappa[m]$. Keys are rerandomized sequentially and at regular intervals. That is, key $\kappa[1]$ is randomized, then after d time units, $\kappa[2]$, then after another d time units, $\kappa[3]$, and so forth—with wraparound. So the DK is completely rerandomized after every dm time units.

We prove that for a natural class of timelines (and the right setting of d), US is the best possible DK scheme, in the sense that it is *undominated*. No other DK scheme achieves better verifier detection probabilities against an optimal adversary across all timelines in the class. As we show, US is undominated for both *prevention* and *detection* timelines.

To develop DK schemes that are usable in practice we must accommodate devices with highly constrained transmission bandwidth. A good choice of m in a US scheme may yield a DK too long for full transmission. As we'll see, our canonical device, an authentication token, can emit at most a few bits of DK data per transmission (per passcode, that is).

In Section V, therefore, we describe an approach to *partial* DK transmission by means of what we call *views*. Views are constructed using a coding scheme that multiplexes a DK over device emissions, essentially like an error-correcting code. Use of views has three benefits, enabling: (1) A long DK to be synchronized with the verifier using short emissions; (2) Resilience to message loss on the device-to-verifier channel; and (3) Clone detection even if an attacker eavesdrops a limited number of times e during the interval $[A, C]$.

Finally, while DKs suit many devices, we explore their application to a particular one: One-time passcode authentication tokens. In Section VI, we show how DKs can be integrated into such tokens without changing their use or passcode format. We briefly describe solutions to several technical challenges, including adversarial tampering and benign user transcription errors. By simulation over authentication-token data from an enterprise with nearly 56,000 unique users, we demonstrate the potential effectiveness of DKs in real-world use.

III. Formal Model

In this section, we lay out a formal security model for Drifting Keys. We first define a *timeline*, the set of time intervals between events in an attack. Then we enumerate the functions composing a DK scheme. Finally, we specify the security of a DK scheme in a standard cryptographic way, in terms of an adversarial experiment.

For conciseness, we now denote a device by $\mathcal{D}$, a verifier by $\mathcal{V}$, and an adversary by Adv where appropriate.

A. Attack timeline

Recall the sequence of event times A, B, and C from above, corresponding respectively to Adv's breach, authentication by $\mathcal{D}$ to $\mathcal{V}$, and Adv's DK forgery attempt.

We let $\Delta = C - A$ denote the length of the entire interval of an attack by Adv. The attack is also characterized by two sub-intervals: The time between the compromise and device authentication, $\Delta_0 = B - A$, and the time between authentication and Adv's forgery, $\Delta_1 = C - B$. Thus $\Delta = \Delta_0 + \Delta_1$.

We refer to the tuple $(\Delta, \Delta_0, \Delta_1)$ as an attack *timeline*. Figure 2 depicts the anatomy of a timeline.

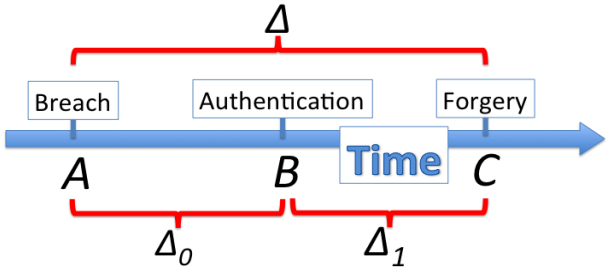

Fig. 2. An attack timeline.

In general, $\mathcal{V}$'s probability of success in detecting Adv's forgery at time C grows as Δ and Δ_0 do. As Δ grows, i.e., more drift takes place, so does Adv's uncertainty about the state of κ at time C. Similarly, as Δ_0 grows (for a fixed Δ), the uncertainty of the verifier about the correct value of κ_C (the range R) diminishes. That is, $\mathcal{V}$'s view of κ is more recent for larger Δ_0—or equivalently, smaller Δ_1.

For successful clone detection, it's a requirement that $\Delta > 0$. If Adv attempts a forgery immediately, i.e. $\Delta = 0$, then Adv has complete knowledge of the current key $\kappa_A = \kappa_C$, and can impersonate the device perfectly.

Similarly, it's a requirement that $\Delta_0 > 0$. If $A = B$, i.e., $\Delta_0 = 0$, then at time C, the verifier and Adv have equivalent knowledge of the current value of κ. In this case Adv can successfully forge with probability 1. (Put another way, in Fig. 1, if $A = B$, then Adv knows R exactly.)

We analyze the security of a given DK scheme below over a family of timelines meeting the condition $\Delta, \Delta_0 > 0$.

Remark: In real-world attack scenarios, typically Adv is constrained to meet some minimum $\Delta > 0$, i.e., delay its forgery attempt for a time after a breach. Often an adversary proceeds

gradually to evade detection, as in Advanced Persistent Threats (APTs) [5], or incurs delays due to the challenge of attacking multiple systems. (For example, an attack against Lockheed-Martin purportedly resulted from the breach of RSA, but surfaced only months later [6].)

It's also common that $\Delta_0 > 0$, meaning that $\mathcal{D}$ was used at some point after the breach, but before Adv's forgery attempt.

B. DK functions

We model time in discrete *timesteps*, e.g., minutes. The current timestep is denoted by t. Values include:

- κ_t: The DK of the device at time t. κ_t is composed of a sequence of m individual sub-keys. We let $\kappa_t[i]$ denote the i^{th} such key, and K denote the space from which individual keys are drawn, i.e., $\kappa_t[i] \in K$, and $c = |K|$. Note that keys may be short, and in some cases may even be single bits, as in Example 1. For convenience, we refer to sub-keys themselves simply as "keys."
- λ_t: The *knowledge* held by the verifier $\mathcal{V}$ of κ_t at time t, where $\lambda_t \in L$ for some space L. For convenience, we assume that t is stored as part of λ_t.
- μ_t: The key-update message generated by the device in the current timestep.

Functions applied by $\mathcal{D}$ and $\mathcal{V}$ include:

- $\mathsf{keyGen}(\ell) \xrightarrow{R} (\kappa_0, \lambda_0)$: A *key generation* function that yields an initial device key and its counterpart for the verifier; here, ℓ is a security parameter that controls m and c. Where m and c are fixed, we omit the use of ℓ.
- $\mathsf{evolve}(t, \kappa_{t-1}) \rightarrow \kappa_t$: A (probabilistic) *key-update function*;
- $\mathsf{updateGen}(t, \kappa_t) \rightarrow \mu_t$: A function that computes an update message;
- $\mathsf{sync}(t, \lambda_s, \mu_t) \rightarrow \lambda_t$: A *server-knowledge update function* that takes in an old state λ_s from time s and updates it with fresh data μ_t at time t.
- $\mathsf{keyVer}(t, \tilde{\kappa}_t, \lambda_s) \rightarrow \{\mathsf{accept}, \mathsf{reject}\}$: A key verification function: Indicates whether the key $\tilde{\kappa}_t$ is consistent with λ_s in the view of $\mathcal{V}$. We say that $\tilde{\kappa}_t$ is *valid* at time t if $\mathsf{keyVer}(t, \tilde{\kappa}_t, \lambda_s) \rightarrow \mathsf{accept}$.

We let $DK = (\mathsf{keyGen}, \mathsf{evolve}, \mathsf{updateGen}, \mathsf{sync}, \mathsf{keyVer})$ denote a DK scheme.

C. Security experiments

A "prevention" experiment: Formally, now, for a given timeline $(\Delta, \Delta_0, \Delta_1)$, and security parameter ℓ, the security of a DK scheme DK is defined in terms of the adversarial experiment specified in Fig. 3.

In this experiment, we model a strong adversary Adv that chooses the time A of its attack, and sets the device key κ_A and the server knowledge λ_A. The device authenticates at time B, and Adv succeeds if it forges a valid DK, i.e., authenticates successfully, at time C; Adv fails if $\mathcal{V}$ rejects its forged DK. We define the success of Adv as $\mathbf{Succ}^{prevent}_{DK,Adv}[\ell](\Delta, \Delta_0, \Delta_1) = \mathsf{pr}[\mathbf{Exp}^{prevent}_{DK,Adv}[\ell](\Delta, \Delta_0, \Delta_1) \rightarrow \mathsf{accept}]$.

Experiment $\mathbf{Exp}^{prevent}_{DK,Adv}[\ell](\Delta, \Delta_0, \Delta_1)$:
Set event times:
$(A, \kappa_A, \lambda_A) \leftarrow Adv(\text{"initialize"})$;
$B \leftarrow A + \Delta_0$;
$C \leftarrow B + \Delta_1$;
Key evolution:
for $t = A + 1$ to B
$\kappa_t \leftarrow \mathsf{evolve}(t, \kappa_{t-1})$;
Set server state:
$\lambda_B \leftarrow \mathsf{sync}(B, \lambda_A, \kappa_B)$;
Adv **tries to guess valid key:**
$\tilde{\kappa}_C \leftarrow Adv(\text{"forge"})$;
Check key validity:
return $\mathsf{keyVer}(C, \tilde{\kappa}_C, \lambda_B)$;

Fig. 3. Basic "prevention" security experiment

This basic experiment models "*prevention*" of an adversarial authentication attempt after an attack, meaning that the verifier detects the forgery attempt immediately. (In practice, the verifier might not actually prevent the authentication, for various reasons – for example, the verifier might not know whether the inconsistent DK came from the device or the attacker.)

In this experiment, Adv doesn't aim to evade eventual clone *detection* completely. In particular, $\mathcal{D}$ could at some time after C send a DK that is inconsistent with Adv's forgery $\tilde{\kappa}_A$. In that case, $\mathcal{V}$ will learn that an attack has occurred, but it will be after Adv's successful authentication.

A "basic detection" experiment: As observed above, $\mathbf{Exp}^{prevent}$ models prevention of authentication by Adv after an attack. Detection after the fact is still of interest, though, in real-world settings. Thus, we also consider a "basic detection" experiment $\mathbf{Exp}^{detect}$ with a ACB sequence of events, in which Adv's forgery attempt takes place before $\mathcal{D}$'s authentication to the verifier.

In this experiment, Adv's breach occurs at time A, Adv forges at time $B = A + \Delta_0$, and $\mathcal{D}$ next authenticates at time $C = B + \Delta_1$. Then Adv is successful if the verifier accepts both Adv's forged key $\tilde{\kappa_B}$ and the device key κ_C as valid. We define the success of a "basic detection" adversary Adv as $\mathbf{Succ}^{detect}_{DK,Adv}[\ell](\Delta, \Delta_0, \Delta_1) = \mathsf{pr}[\mathbf{Exp}^{detect}_{DK,Adv}[\ell](\Delta, \Delta_0, \Delta_1) \rightarrow \mathsf{accept}]$. A formal specification of the basic detection experiment is given in the full version of this paper [**?**].

A "full detection" experiment: Since a forgery attempt by Adv's may be detected by $\mathcal{V}$ via inconsistency with $\mathcal{D}$'s authentication either before or after the forgery, we consider a "full detection" experiment $\mathbf{Exp}^{full}$ where $\mathcal{D}$ authenticates both before and after Adv's authentication attempt.

In this experiment, Adv's breach occurs at time A, $\mathcal{D}$ authenticates at time $B = A + \Delta_0$, Adv forges at time $C = B + \Delta_1$, and $\mathcal{D}$ authenticates again at time $D = C + \Delta_2$. Then Adv succeeds if the verifier accepts both Adv's forged key $\tilde{\kappa_C}$ and $\mathcal{D}$'s key κ_D, meaning that $\tilde{\kappa_C}$ is consistent with both κ_B and κ_D. We define the success of a "full detec-

tion" adversary Adv as $\mathbf{Succ}^{full}_{DK,Adv}[\ell](\Delta,\Delta_0,\Delta_1,\Delta_2) = \mathsf{pr}[\mathbf{Exp}^{full}_{DK,Adv}[\ell](\Delta,\Delta_0,\Delta_1,\Delta_2) \to \mathsf{accept}]$. A formal specification of the full detection experiment is given in the full version of this paper [**?**].

Other variations: For DK schemes we describe later in which $\mathcal{D}$ transmits only partial "views" of κ, experiments must include multiple $\mathcal{D}$-$\mathcal{V}$ synchronizations.

IV. Uniformly Staggered (US) Updates

In this section, we consider a DK scheme in which each of $\mathcal{D}$'s m keys is randomized periodically and sequentially. Each key has the same period p but a distinct phase that is a multiple of d for $d = p/m$ (assuming here that $m \mid p$).

More formally, each key $\kappa_t[i]$ has an associated phase $d_i = d \cdot i$, and key updates proceed according to the following rule:

- *Key-Update Rule:* Update key $\kappa_t[i]$, i.e., set $\kappa_t[i] \xleftarrow{R} K$, at time t if $t = d_i \pmod p$. Otherwise, $\kappa_t[i] \leftarrow \kappa_{t-1}[i]$, i.e., the key remains unchanged.

We call this a *Uniformly Staggered* (US) scheme with period p and phase shift d. Its use is illustrated in this example:

Example 2: A hardware token contains a DK κ_t consisting of seven one-bit keys, each updated on a different day of the week. (It maintains this key in addition to its primary key for generating passcodes.) The first bit is randomized every Sunday, the second every Monday, and so forth. That is, it employs US with $p = 7$ and $d = 1$ (where timesteps are days). Thus $m = p/d = 7, \kappa_t = \{\kappa_t[1], \ldots, \kappa_t[7]\}$, $K = \{0,1\}$, and key $\kappa[i]$ has $(p_i, d_i) = (7, i)$ for $1 \le i \le 7$.

Each displayed / transmitted passcode has the current DK embedded in it. (See Section VI for details.)

A. Prevention and detection probabilities for US

We now give the prevention and detection probabilities for US with period $p = md$ and phase shift d over key space K^m, where $|K| = c$. For a timeline $(\Delta, \Delta_0, \Delta_1)$ with $\Delta_0 + \Delta_1 = \Delta \le md$, an optimal $\mathbf{Exp}^{prevent}$ adversary and an optimal $\mathbf{Exp}^{detect}$ adversary succeed with probability $\mathbf{Succ}^{prevent}_{\mathrm{US},Adv}(\Delta,\Delta_0,\Delta_1) = \mathbf{Succ}^{detect}_{\mathrm{US},Adv}(\Delta,\Delta_0,\Delta_1) = c^{-\lfloor \Delta_0/d \rfloor}$. Thus, the probability of prevention or detection of Adv's forgery is $1 - c^{-\lfloor \Delta_0/d \rfloor}$.

For a timeline $(\Delta, \Delta_0, \Delta_1, \Delta_2)$ with $\Delta_0 + \Delta_1 + \Delta_2 = \Delta \le md$, an optimal $\mathbf{Exp}^{full}$ adversary succeeds with probability $\mathbf{Succ}^{full}_{\mathrm{US},Adv}(\Delta,\Delta_0,\Delta_1,\Delta_2) = c^{-\lfloor (\Delta_0+\Delta_1)/d \rfloor}$. Thus, the probability that Adv's forgery is detected (either immediately or when $\mathcal{D}$ authenticates later) in $\mathbf{Exp}^{full}$ is $1 - c^{-\lfloor (\Delta_0+\Delta_1)/d \rfloor}$.

B. Security analysis: Prevention

We now show that for a broad "symmetric" class of prevention timelines in which $\Delta = md$, US is *undominated*. A scheme is undominated if there exists no scheme that dominates it in the sense of achieving better detection across all timelines. Define

$$\vec{\mathbf{\Delta}}_{m,d} = \{\mathbf{\Delta} = \{(\Delta = md, \Delta_0, \Delta_1) \mid (\Delta_0, \Delta_1) \in \{(s, md-s), (md-s, s)\}\}_{s \in S}\}_{S \subseteq [0, md]}. \quad (1)$$

Note that for any $\mathbf{\Delta} \in \vec{\mathbf{\Delta}}_{m,d}$, if $(\Delta, \Delta_0, \Delta_1) \in \mathbf{\Delta}$, then $(\Delta, \Delta_1, \Delta_0) \in \mathbf{\Delta}$. Also note that the class $\vec{\mathbf{\Delta}}_{m,d}$ includes, e.g., the set of *all* timelines in which $\Delta = md$. Also define:

- $\mathsf{R}(t, \kappa_t, \delta)$: The *drift range* of a key κ_t over an interval of time. Given κ_t, $\mathsf{R}(t, \kappa_t, \delta)$ is the set of possible values assumed by $\kappa_{t+\delta}$.

We prove the following theorem:

Theorem 1: Given key space K^m, US with period $p = md$ and phase shift d is undominated with respect to $\mathbf{Exp}^{prevent}_{DK,Adv}$ for any $\mathbf{\Delta} \in \vec{\mathbf{\Delta}}_{m,d}$. That is, $\max_{Adv,(\Delta,\Delta_0,\Delta_1) \in \mathbf{\Delta}} \mathbf{Succ}^{prevent}_{DK,Adv}(\Delta,\Delta_0,\Delta_1)$ is minimized by $DK = \mathrm{US}$.

Proof: For simplicity, assume all intervals below are multiples of d. In US, for any t, κ_t, and $\delta \le md$, we have $|\mathsf{R}(t, \kappa_t, \delta)| = c^{\delta/d}$. So for any $(\Delta, \Delta_0, \Delta_1)$ with $\Delta = \Delta_0 + \Delta_1$, there exists an Adv for which $\mathbf{Succ}^{prevent}_{\mathrm{US},Adv}(\Delta,\Delta_0,\Delta_1) = c^{-\Delta_0/d}$. This Adv simply guesses a key $\kappa'_B \in \mathsf{R}(A, \kappa_A, \Delta_0)$ and then sends an arbitrary key $\kappa'_C \in \mathsf{R}(B, \kappa'_B, \Delta_1)$. If its guess κ'_B is correct, then $\mathcal{V}$ will accept κ'_C.

Suppose there exists a drifting-key scheme $\tilde{DK}$ and pair $(\tilde{\Delta_0}, \tilde{\Delta_1})$ such that for any Adv (choosing any attack time $\tilde{A}$), $\mathbf{Succ}^{prevent}_{\tilde{DK},Adv}(\Delta,\tilde{\Delta_0},\tilde{\Delta_1}) < c^{-\tilde{\Delta_0}/d}$. In other words, $\tilde{DK}$ provides stronger security than US for a particular timeline $(\Delta, \tilde{\Delta_0}, \tilde{\Delta_1})$.

Then for any $\tilde{A}$ and $\kappa_{\tilde{A}}$, there exists at least one key $\kappa_{\tilde{B}} \in \mathsf{R}(\tilde{A}, \kappa_{\tilde{A}}, \tilde{\Delta_0})$ such that $|\mathsf{R}(\tilde{B}, \kappa_{\tilde{B}}, \tilde{\Delta_1})| < c^{m-\tilde{\Delta_0}/d}$. (Otherwise, Adv can simply guess a random key $\kappa'_{\tilde{C}} \in K^m$; this key will be valid with probability $\ge c^{m-\tilde{\Delta_0}/d}/c^m = c^{-\tilde{\Delta_0}/d}$.) Let ν denote one such key $\kappa_{\tilde{B}}$. We may view ν as inducing "slow" key-space growth over the interval $[\tilde{B}, \tilde{C}]$.

Then Adv can take advantage of the slow key-space growth that ν induces over interval $[\tilde{B}, \tilde{C}]$ in the prevention experiment for a different timeline in $\mathbf{\Delta}$, namely $(\Delta, \Delta_0 = \tilde{\Delta_1}, \Delta_1 = \tilde{\Delta_0})$. In $\mathbf{Exp}^{prevent}_{\tilde{DK},Adv}$, Adv sets $A \leftarrow \tilde{B}$, $\kappa_A \leftarrow \nu$, and $\lambda_A \leftarrow (\tilde{B}, \nu)$.

Now Adv's key-guessing strategy is as follows. It picks a random key κ'_B in $\mathsf{R}(A, \kappa_A, \Delta_0)$. It guesses for κ'_C in $\mathbf{Exp}^{prevent}_{\tilde{DK},Adv}$ an arbitrary key consistent with κ'_B, i.e., $\kappa'_C \in \mathsf{R}(B, \kappa'_B, \Delta_1)$.

Clearly, if $\kappa'_B = \kappa_B$, i.e., Adv correctly guessed the key at time B, then $\mathcal{V}$ will accept κ'_C as valid. Since $|\mathsf{R}(A, \kappa_A = \nu, \Delta_0)| < c^{m-\tilde{\Delta_0}/d}$, Adv will thus succeed with probability $> c^{-(m-\tilde{\Delta_0}/d)} = c^{-(m-(md-\Delta_0)/d)} = c^{-\Delta_0/d}$. Thus, $\mathbf{Succ}^{prevent}_{\tilde{DK},Adv}(\Delta,\Delta_0,\Delta_1) > \mathbf{Succ}^{prevent}_{\mathrm{US},Adv}(\Delta,\Delta_0,\Delta_1)$.

Since no $\tilde{DK}$ dominates US, US is undominated. ∎

C. Security analysis: Detection

We have proven that US is undominated with respect to prevention for a natural class of timelines. Using similar but somewhat more involved arguments, we prove that US is also undominated with respect to *basic detection*, for the same class of timelines, in the following theorem:

Theorem 2: Given key space K^m, US with period $p = md$ and phase shift d is undominated with respect to $\mathbf{Exp}^{detect}_{DK,Adv}$ for any $\mathbf{\Delta} \in \vec{\mathbf{\Delta}}_{m,d}$. That is,

$\max_{Adv,(\Delta,\Delta_0,\Delta_1)\in\boldsymbol{\Delta}} \mathbf{Succ}^{detect}_{DK,Adv}(\Delta,\Delta_0,\Delta_1)$ is minimized by $DK = \text{US}$.
The proof can be found in the full version of the paper [**?**].

We conjecture that US is also undominated with respect to *full detection* for a class of timelines satisfying some "symmetry" condition.

Conjecture 1: Given key space K^m, US with period $p = md$ and phase shift d is undominated with respect to $\mathbf{Exp}^{full}_{DK,Adv}$ for any $\boldsymbol{\Delta} \in \vec{\boldsymbol{\Delta}}'_{m,d}$ for some natural class $\vec{\boldsymbol{\Delta}}'_{m,d}$. That is, $\max_{Adv,(\Delta,\Delta_0,\Delta_1,\Delta_2)\in\boldsymbol{\Delta}} \mathbf{Succ}^{full}_{DK,Adv}(\Delta,\Delta_0,\Delta_1,\Delta_2)$ is minimized by $DK = \text{US}$.

V. Partial DK Transmission via "Views"

We now consider an approach to DKs in which at time t, the device transmits a *partial* DK, rather than its complete DK κ_t. We call this partial DK as a *view*. As we show, a view can be arbitrarily compact—even just a single bit.

View transmission has three advantages over full-DK transmission: (1) An Adv that intercepts a view doesn't learn the full DK state κ_t of the device, so views create resilience to eavesdropping; (2) Transmitting a view is less bandwidth-intensive than full DK transmission; and (3) Transmitting views creates resilience to a noisy channel, in particular, an erasure channel, i.e., one that drops symbols.

These advantages are particularly valuable for authentication tokens, our archetypal device example. These devices are highly bandwidth-constrained, as explained above. Moreover, their passcodes are transmitted over what may be viewed as a *very* noisy erasure channel. The vast majority of passcodes generated by a token aren't actually transmitted. Only when a user authenticates does she transcribe one.

The one drawback to views is that they require a verifier to collect multiple views in order to achieve the same probability of clone detection as a full-DK transmission scheme.

We construct partial DKs for transmission by means of an encoding scheme whose structure we now present. This scheme is essentially a *convolutional code* [22], an error-correcting code that operates over message data streaming through a fixed size buffer (the DK, in our case). Our main interest here, however, is error detection (cloning), rather than error correction. Additionally, decoding efficiency is an important design goal.

A. Coding framework

Given a DK κ_t, the aim of a DK coding scheme is to produce a sequence of (compact) views for device-to-verifier transmission. The verifier should be able to detect views output by incompatible key sources. That is, given views from a key sequence $\kappa_1,\ldots,\kappa_t$ and from a (clone) key $\tilde{\kappa}_t$, the verifier should detect an inconsistency in views with high probability.

A DK coding scheme operates over a set of information elements called *symbols*, denoted by Σ. Additionally, it makes use of a *counter* value c on the number of views generated during a key update period t.

There are two functions in a DK coding scheme:

- $\mathsf{encode}(\kappa_t, t, c, [K]) \to \sigma \in \Sigma$: Encoding operates over a DK κ_t consisting of a sequence of k symbols, the current time, and a counter value $c \in Z^+$. It outputs a single information symbol σ. (Our main proposed scheme employs an additional input key K.)
- $\mathsf{verify}(\{(\sigma_j, t_j, c_j)\}_{j=1}^q, [K]) \to \{\mathsf{accept}, \mathsf{reject}\}$: The verify function takes as input a set of q views and associated times and counters. It accepts the set if it represents a sequence of views over some valid sequence of keys $\kappa_1,\ldots,\kappa_t$. Otherwise it rejects. (Our main proposed scheme employs an additional input key K.)

A simple example illustrates the basic operation of a DK coding scheme.

Example 3: In a naïve, example DK coding scheme, $\mathsf{encode}(\kappa_t, t, c)$ *simply outputs* $\kappa_t[c \bmod k]$*, i.e., outputs raw key values (with counter wraparound).*

In this case, verify *accepts a set* $\{(\sigma_j, t_j, c_j)\}_{j=1}^q$ *of views provided that there are no inconsistent values for any key symbol* $\kappa_t[i]$*. That is, it accepts iff there exist no two triples* (σ_j, t_j, c_j) *and* $(\sigma_{j'}, t_{j'}, c_{j'})$ *with* $j \neq j'$ *such that* $t_j = t_{j'}$*,* $c_j = c_{j'} \bmod k$*, and* $\sigma_j \neq \sigma_{j'}$*.*

If we regard freshly generated key symbols as message insertions into a buffer (the current key), encode operates essentially like the encoder in a convolutional error-correcting code, while verify performs error-detection on code output.

B. A coding scheme

We now describe a simple, practical coding scheme that we call a *dot-product* scheme. This scheme is attractive for highly constrained environments in which key symbols are bits, i.e., $\Sigma = \{0,1\}$. We employ this scheme in our authentication-token proposed DK construction.

Let $\vec{v}_{t,c} \in \{0,1\}^k$ be a uniformly random bit-vector of length k. (These vectors may be derived jointly by the device and verifier from a pseudorandom number generator applied to shared key K.) A view, then, is a single bit, the dot product of a vector with its corresponding key, i.e.:

- $\mathsf{encode}(\kappa_t, t, c, K) \to \kappa_t \cdot \vec{v}_{t,c} \in \{0,1\}$.

Verification involves a consistency check on the set of linear equations implied by views, namely:

- $\mathsf{verify}(\{(\sigma_j, t_j, c_j)\}_{j=1}^q, K)$ outputs accept if, for $t = \max_{j=1}^q t_j$, there exists a solution $\kappa_1,\ldots,\kappa_t$ to the set of linear equations $\{\kappa_{t_j} \cdot \vec{v}_{t_j,c_j} = \sigma_j\}_{j=1}^q$. Otherwise it outputs reject.

This dot-product coding scheme, like the naïve one of Example 3, is an error-detecting code. But in our dot-product DK code, *views are a function of all bits*. Consequently, given an adversary Adv with an incorrect key $\tilde{\kappa}_t \neq \kappa_t$, the verifier can detect a cloning attempt with high probability irrespective of where the erroneous bits lie in $\tilde{\kappa}_t$.

Randomization of $\vec{v}_{t,c}$ allows the verifier to solve for erasures, i.e., key bit values erased by the noisy channel, as it receives transmissions from the device.

As a heuristic optimization for our authentication-token application below, we fix certain bits of $\vec{v}_{t,c}$. For example,

when there's only one view per passcode, we set $\vec{v}_{t,c}[f] = 1$, for f the (unique) freshest updated bit position. A view then *always* captures the freshest drifting bit and if Adv doesn't know this bit, it will guess the view incorrectly with probability at least $1/2$, resulting in a high detection rate.

VI. Application to Authentication Tokens

We now consider DKs in our archetypal application, authentication tokens. There are a number of such tokens that operate in more or less the same way, displaying decimal-digit passcodes for transcription. SecurID and Google Authenticator are two prominent examples.

We explore the application here of a bit-based US scheme; again, this means bits are randomized sequentially (with wraparound). For concreteness, we consider $k = 7$ DK bits. (We vary other parameters in our experiments.) We consider eight-digit passcodes (a standard SecurID option).

A. Embedding in Passcodes

To create a legacy-compatible system—or simply not make passcodes longer—it's necessary to *embed* a DK channel within token passcodes. (Note, however, that a small sacrifice of authentication strength results: Each bit devoted to the DK channel is one fewer bit in the authentication channel.)

Consider, for concreteness, the use of two views, yielding dot-product encoded bits (a, b) for a given passcode. A naïve embedding approach is to replace the last two bits of the passcode with these DK-channel bits (a, b). Two problems then arise:

1) *Adversarial DK alarms:* A man-in-the-middle adversary Adv, one capable of reading and tampering with P in transmission, can extract and modify DK values—without even compromising the token. Adv can create false alarms in the DK system.[1]
2) *User-induced false DK alarms:* Benign mistyping of a passcode by a user can, with high probability, change (a, b) while leaving the rest of P intact, resulting in a false alarm in the DK system.

To address these problems, a DK channel may be created by means of a secret encoding of (a, b) with redundancy. Lacking space to explore such encodings at length, we just give an example scheme that seems attractive for practice. Let P denote the passcode for a given time t and Q denote the *DK-enhanced* passcode yielded by embedding view bits.

Example 4 (DK-channel encoding scheme): For each authentication token, a distinct key is selected consisting of two pairs of distinct, randomly selected digits $(x_0, x_1), (y_0, y_1) \in_R \{0, 1, \ldots, 9\}^2$.

At a given time t, the pair of bits (a, b) *is mapped into an eight-digit codeword* $C = x_a x_a x_a x_a y_b y_b y_b y_b$.

The DK-enhanced passcode is computed as $Q \leftarrow P \oplus C$. *Here* $\oplus$ *denotes digitwise, mod 10 addition. The server accepts a passcode* Q' *as valid if* $Q' \oplus P$ *is a valid codeword* C. *(The server then extracts* (a, b) *from* C.*)*

To modify (a, b) *without rendering* Q *invalid, the attacker must guess* $x_0 - x_1$ *or* $y_0 - y_1$, *which it can do successfully with probability at most* $1/10$.

For a user to mistype a passcode Q *and change* (a, b) *without rendering the passcode invalid would require four mistyped digits in sequence—a highly improbable event.*

Of course, many different embedding schemes are possible, with various technical tradeoffs.

B. Simulation

To evaluate the efficacy of views of a bit-based US scheme for authentication tokens, we simulate a powerful attacker and study how different parameter choices impact the probability of detecting an attack and the average time until detection. Our simulated users consist of nearly 56,000 unique authentication tokens that were used to login during a 3-month window at a large enterprise. Our dataset contains over 4 million unique logins collected from March 22, 2012 to June 19, 2012. We find that on average tokens are used three times a week, but with high variability of inter-login intervals. We present more details about the user population in the full version of the paper.

The set of tokens in our dataset isn't static; new tokens are added and old tokens retired. Having no way of distinguishing users who went on vacation and didn't log in from those users who left the group and no longer have login privileges, we treated all records as belonging to permanent users. (New and terminated users skew our results pessimistically, in the sense of creating the appearance of longer inter-login intervals and making attacker detection harder.)

We simulate attack against each of our 56,000 users and repeat each test 10 times for each parameter setting. In each run the attacker randomly selects a time to break into the token, and must then wait at least Δ_{min} time before submitting a forged passcode, where Δ_{min} is parameterized on a per-experiment basis as a multiple (or fraction) of the drift frequency. If Δ_{min} is less than 1, it's possible that no bits will have drifted before the attack, in which case the forgery will always succeed. If bits have drifted, the attacker must guess any unknown bits corresponding to the current views. The attacker in our simulations, though, is powerful, in the sense that he can not only compromise a user's token at will, but may also launch his attack advantageously at a time when the views require the attacker to guess as few bits as possible.

Using the heuristic mentioned above, views in our simulated tokens are crafted to ensure transmission of recently drifted bits. In particular, if w views are transmitted per passcode, then the first vector terminates with a '1' bit, the next with a '10,' etc., through '10^w'; all other vector bits are randomly selected, as in our basic dot-product scheme. As a result of this view composition, the attacker cannot select a time in which views omit the w most recent drift bits, but can wait until views omit all of the other drift bits that are unknown

[1] An adversary can instigate repeated false alarms in order to erode administrator sensitivity to signals of a real intrusion.

to him. The time the attacker must wait for such a favorable scenario is not prohibitive for the practical parameterizations we consider in our experiments. (For example, given $w = 1$, and 7 drift bits, the attacker need wait only 32 minutes on average for the first six view bits to assume a desired value.)

Such adversarial waiting limits the best possible detection probability. The attacker must submit guesses for the w most recent drift bits; if fewer than w bits have drifted, however, the attacker will know the value of those remaining bits. Each bit drifts independently, so the probability of the attacker correctly guessing a given drift bit is $\frac{1}{2}$. If d is the number of bits that have drifted since the attacker broke in, ($d \geq \Delta_{min}$), then the number of bits the attacker has to guess is $g = min(w, d)$. The ideal detection probability then is $1 - (\frac{1}{2})^g$.

If the server and token are in sync at the time of the attack, then the ideal detection probability is actually a prevention probability. However, if the attacker is the first to submit a passcode after a bit drifts, even if he guesses wrong, the inconsistency cannot be detected until the user submits a passcode. In fact it may take several passcode transmissions before the inconsistency can be detected.

In our simulation, an individual user submits passcodes at times corresponding to those for the user in the log. The server and token are assumed to be in sync at the beginning of the window (March 22), but drift and re-sync as time passes and users submit passcodes. We measure the percentage of attacks that are prevented (detected immediately upon submission by the adversary), those that are eventually detected and how long such detection takes, and those for which the server is unable to catch up and detect the attack in the remaining portion of the three-month simulation window.

As Figure 4 shows, the longer the attacker is forced to wait between breaking into the token and submitting a forged passcode, the higher the probability of detection. Likewise, and more importantly, the more bandwidth per passcode (number of views), the higher the detection probability and the quicker the detection (from 36% detection in an average of 56 hours for $w = 1$ view bits to 78% detection in an average of 20 hours with $w = 3$ view bits for $\Delta_{min} = 3$ and a drift frequency of 3 days). The reason for this is two-fold. First, the attacker must guess more bits if multiple bits have drifted since the break-in. Second, the additional information provided in each passcode helps the server stay in sync, increasing both the chances of prevention and speeding up eventual detection. There are other ways to increase view bandwidth, such as having users submit passcodes more often, or submit multiple passcodes if they login less frequently. We leave analysis of this and other protocol-level changes as future work.

As noted above, users in our dataset log in on average only three times a week. When bits drift every week, each new bit will be sampled three times, on average, allowing the server to stay in sync. Bits that drift every 3 days more often go unsampled, slowing or even preventing detection of attacks.

Remark: In our experiments, DKs don't detect attacks with overwhelming probability. That isn't the goal here or in general. Rather, it's to provide successful prevention or detection with *high* probability. We advocate DKs particularly for detection of *systemic* attacks, such as server-side breaches or attacks on multiple devices. In such cases, the detection rates we observe in our experiments here (18%-85%) are appropriate. Given a substantial number of compromised devices or attacker events, systemwide detection rates can approach 100%. We also underscore that *without DKs, full compromise of cryptographic keys and cloning are generally undetectable*.

VII. Related Work

Tamper-evident signatures, proposed by Itkis in [10], are closely related to DKs. A tamper-evident signing key evolves randomly: An old (private) key is periodically used to sign a new (public) key. As with DKs, signatures forged by the attacker after device compromise will, with high probability, diverge from those of the device, leading to verifier detection.

Tamper-evident signatures are stronger than DKs because signature forgery is infeasible, but far more resource-intensive, and unsuitable for constrained devices. In Itkis's scheme, the public key grows over time as does the required number of digital signature operations. The scheme also can't support DK-type views, so reliable transmission of public keys requires high bandwidth. Itkis's work is the closest example to DK of a key-evolving cryptographic scheme; see [8] for a broad survey. Generally, key-evolving schemes aim to protect past or future keys, not to detect key compromise.

As DKs assume unidirectional communication from the device to verifier, they may be seen as a simple form of *unidirectional key distribution* (plus authentication) designed for lossy networks. "Self-healing key distribution" [17], [25] initiated this line of research. It involves broadcast transmission of full key sets, however, and is designed for a group manager to distribute session keys, not for cloning detection.

Unidirectional key distribution is also used in the resource-constrained context of RFID to enforce privacy [11]. Amariucai et al. [1] propose its use for device pairing in a scheme based on leakage of and subsequent cracking of a secret key. We note that DK views are actually a more efficient, easier-to-analyze way of gradually distributing a key unidirectionally, and thus can serve as an improvement over [1].

Clone detection for highly constrained devices is a problem of great interest for sensor networks and RFID-enabled supply chains. In sensor networks, nearly all proposed countermeasures rely on cooperative protocols among nodes, e.g., [3], [4], [19]. DKs, in contrast, don't rely on networking of devices.

Common barcode-type RFID tags are easily cloned [14]. But RFID-enabled supply chains are highly structured: Goods and tags travel along a small set of pre-defined paths. Several schemes thus rely on divergent physical tag paths to detect cloning and other attacks, e.g.,[7]. Similar in spirit to DKs is [15], which rotates a secret value in tags; each tag has just one value, though, rotated in its entirety by a centralized entity.

We believe that DKs should be of interest as a new, lightweight approach to clone detection in both sensor networks and RFID-enabled systems.

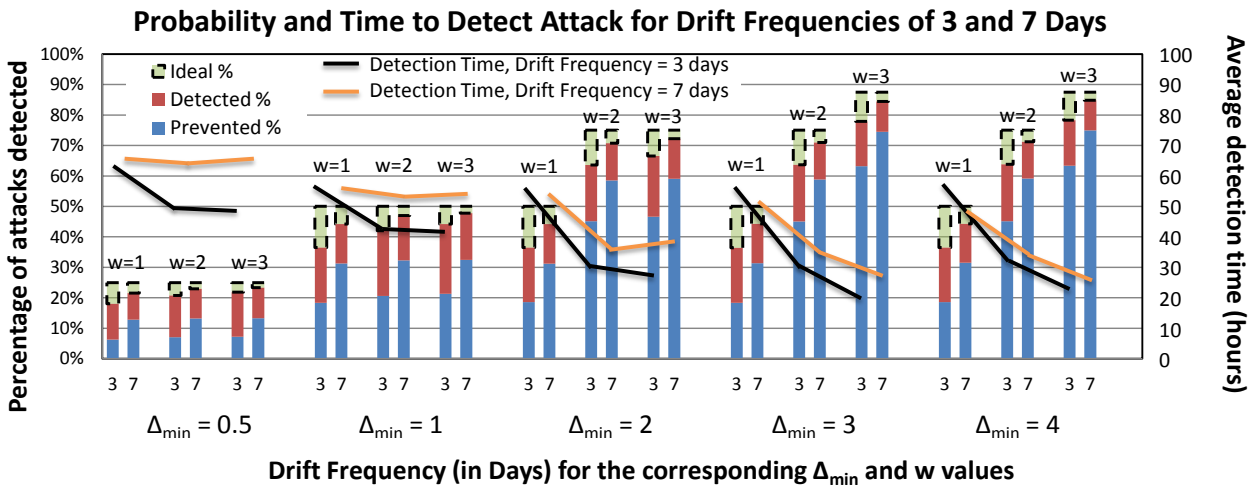

Fig. 4. Prevention and Detection Probability, and Average Time to Detection from simulation of token with 7 drifting bits which drift every 3 or 7 days.

VIII. Conclusion

Resource-constrained devices, such as sensors, authentication tokens, and RFID tags are increasingly pervasive sources of high-value data. Their vulnerability to complete compromise of secret state locally and through server-side breaches highlights the importance of protections against cloning attacks. Practical cloning detection is challenging, though, for devices with low bandwidth, power, and computational ability.

We have introduced Drifting Keys as a lightweight approach to this challenge. DK emissions can be made arbitrarily compact for bandwidth constrainted devices, are resilient to lossy networks, and require no server-to-device communication. The simplicity of DKs lends them to analytic study, as shown by our security proofs showing that Uniformly Staggered (US) DK schemes are undominated over a natural class of timelines.

Additionally, our exploration of one-time passcode authentication tokens has shown the practical promise of DKs. DKs can be embedded in one-time passcode authentication tokens with no modification to passcode structure and little resource overhead. Our experiments with the authentication logs of a large enterprise bear out DKs' efficacy and practicality.

References

[1] G. T. Amariucai, C. Bergman, and Y. Guan. An automatic, time-based, secure pairing protocol for passive RFID. In *RFIDSec*, pages 108–126, 2011.

[2] R. Anderson and M. Kuhn. Tamper resistance — a cautionary note. In *USENIX Security*, pages 1–11, 1996.

[3] M. Conti, R. Di Pietro, L. V. Mancini, and A. Mei. Distributed detection of clone attacks in wireless sensor networks. *IEEE Trans. Dependable Sec. Comput.*, 8(5):685–698, 2011.

[4] M. Conti, R. Di Pietro, and A. Spognardi. Wireless sensor replica detection in mobile environments. In *Proceedings of the 13th international conference on Distributed Computing and Networking*, ICDCN'12, pages 249–264, Berlin, Heidelberg, 2012. Springer-Verlag.

[5] Damballa, Inc. Advanced persistent threats. Whitepaper referenced 2012 at bit.ly/eS0B0H, 2010.

[6] C. Drew. Stolen data is tracked to hacking at Lockheed. *New York Times*, page B1, 4 June 2011.

[7] K. Elkhiyaoui, E.-O. Blass, and R. Molva. Checker: on-site checking in RFID-based supply chains. In *Proceedings of the fifth ACM conference on Security and Privacy in Wireless and Mobile Networks*, WISEC '12, pages 173–184, New York, NY, USA, 2012. ACM.

[8] M. Franklin. A survey of key evolving cryptosystems. *International Journal of Security and Networks*, 1(1):46–53, 2006.

[9] D. Goodin. RSA breach leaks data for hacking SecurID tokens. *The Register*, 18 March 2011.

[10] G. Itkis. Forward security: Adaptive cryptography—time evolution. In *Handbook of Information Security*. John Wiley and Sons, 2006.

[11] A. Juels, R. Pappu, and B. Parno. Unidirectional key distribution across time and space with applications to RFID security. In *USENIX Security*, pages 75–90, Berkeley, CA, USA, 2008. USENIX Association.

[12] P.C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In *CRYPTO*, pages 104–113, 1999.

[13] P.C. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In *CRYPTO*, pages 388–397, 1999.

[14] K. Koscher, A. Juels, V. Brajkovic, and T. Kohno. EPC RFID tag security weaknesses and defenses: passport cards, enhanced drivers licenses, and beyond. In *Proceedings of the 16th ACM conference on Computer and communications security*, CCS '09, pages 33–42, New York, NY, USA, 2009. ACM.

[15] M. Lehtonen, F. Michahelles, and E. Fleisch. How to detect cloned tags in a reliable way from incomplete RFID traces. In *IEEE International Conference on RFID*, pages 257–264, 2009.

[16] A. K. Lenstra, J. P. Hughes, M. Augier, J. W. Bos, T. Kleinjung, and C. Wachter. Ron was wrong, Whit is right. Technical Report 2012-064, IACR, 2012.

[17] D. Liu, P. Ning, and K. Sun. Efficient self-healing group key distribution with revocation capability. In *Proceedings of the 10th ACM conference on Computer and communications security*, CCS '03, pages 231–240, New York, NY, USA, 2003. ACM.

[18] Y. Oren and A. Shamir. Remote password extraction from RFID tags. *IEEE Transactions on Computers*, 56(9):1292– 1296, 2007.

[19] B. Parno, A. Perrig, and V. Gligor. Distributed detection of node replication attacks in sensor networks. In *IEEE Security and Privacy Symposium*, pages 49– 63, 2005.

[20] N. Perlroth. Lax security at LinkedIn is laid bare. *New York Times*, page B1, 11 June 2012.

[21] N. Perlroth. Yahoo breach extends beyond Yahoo to Gmail, Hotmail, AOL users. Blog, 12 July 2012.

[22] W.W. Peterson and E.J. Weldon, Jr. *Error-Correcting Codes (2nd edition)*. MIT Press, 1972.

[23] RSA, The Security Division of EMC. RSA hardware authenticators product description. http://www.rsa.com/node.aspx?id=1158, 2012.

[24] N. D. Schwartz and C. Drew. RSA faces angry users after breach. *New York Times*, page B1, 8 June 2011.

[25] J. Staddon, S. Miner, M. Franklin, D. Balfanz, M. Malkin, and D. Dean. Self-healing key distribution with revocation. In *IEEE Symposium on*

Security and Privacy, pages 241–257, Los Alamitos, CA, USA, 2002. IEEE Computer Society.

[26] D. Zanetti, L. Fellmann, and S. Capkun. Privacy-preserving clone detection for RFID-enabled supply chains. In *Proc. IEEE Int. Conf. on RFID*, 2010.