ReDABLS: Revisiting Device Attestation with Bounded Leakage of Secrets

Jun Zhao, Virgil Gligor, Adrian Perrig*, and James Newsome

CyLab and ECE Department
Carnegie Mellon University
Pittsburgh, PA 15213
{junzhao,gligor,perrig,jnewsome}@cmu.edu

Abstract. Many commodity operating systems and applications become infested with malicious software over time, primarily due to exploits that take advantage of software flaws and operator errors. In this paper, we present the salient features of a system design which allows remote-device authentication by a verifier, reaching malware-free system states, and trusted application booting in the presence of malicious software that is controlled by a network adversary. Our system design revisits the notion of *device attestation with bounded leakage of secrets* (DABLS), and illustrates both the significant challenges of making it work in practice and how to overcome them.

1 Introduction

During the past decade, Professor Moriarty, the fictitious genius and evil adversary, has acquired new attack capabilities and now poses an unprecedented challenge for the wizards of the Security Protocols Workshop (SPW). Not only can he fully control communication networks (e.g., in man-in-the-middle style) connecting remote devices with device-attestation hosts, but also he can now inject malware into those devices, making them behave in a Byzantine manner and/or leak their secrets. In the past, the wizards were able to counter either one of these attack capabilities or the other, but not both together.

For example, if Moriarty controls all network communications but not end hosts and devices, the wizards of SPW could deploy secret encryption keys in the commodity cryptographic modules; e.g., the Trusted Platform Modules (TPMs) [10] of remote hosts and devices, and take advantage of Moriarty's bounded computational power to counter his attestation attacks with provable-secure protocols. Or, if he could only insert malware in some of the remote devices but not control any network communications, remote connections to devices could be authenticated without device secrets; e.g., by using network front-ends, which remain beyond the reach of device malware, for remote devices. In this setting, the wizards could deploy sufficiently many additional devices beyond Moriarty's

* Current address: Computer Science Department, ETH, Zurich, Switzerland.

reach to detect Byzantine misbehavior by the malware-infected ones, and enable host reliance only on clean devices. The strategy of relying on secret keys could be safely foregone and the significant challenge of secure, remote key management on malware controlled commodity hardware[1] avoided, in this setting. However, if use of secret keys would still be desired, each device could store a key fragment and rely on threshold cryptography - with the appropriate assumptions – to assemble a shared secret key that Moriarty could not obtain from the set of devices he controls.

Three of the questions faced in designing attestation protocols for remote devices are as follows.

(1) How could a host authenticate a remote device, when adversary Moriarty controls *both* the communication network (i.e., via network adversary M_{out}) and the remote device software (i.e., via device malware M_{in}), but not its commodity hardware (e.g., physical device configuration, components, channel bandwidth)? Clearly, device authentication requires that a device's secret be protected, and if secrets protected by commodity hardware could be discovered by Moriarty's malware M_{in} and exported to his network adversary M_{out}, he could then use his own bogus device to masquerade as the authentic remote device. Could we exploit hardware architecture features to perform software-based device authentication without depending on long-lived, hardware-protected secrets; e.g., without physically unclonable functions (PUFs) [1, 4], TPMs?

(2) How could a remote device prove that it has reached a malware-free state to an attestation host *after* Moriarty has inserted malware M_{in} into that device? Remotely re-booting device software and initializing a malware-free state could not be performed with significant assurance since malware M_{in} itself could compromise the reboot operation. Even if device authentication could be performed in the presence of malware M_{in} (e.g., by using special hardware, such as PUFs), proving the establishment of a malware-free state on a commodity device to an attestation host remains a challenge. Is there a way to clean up the device remotely and eliminate malware M_{in} despite the network adversary M_{out}?

(3) How could a remote device prove that it has performed a *trusted boot* of application software to an attestation host? Notice that between the time that a malware-free device state is demonstrated to an attestation host and the time that trusted boot of application software is completed, the device could be compromised by Moriarty's malware M_{in} again. Could a proof of correct device authentication be composed with one of malware-free state establishment and further with one of trusted boot of application software?

In this paper, we provide preliminary answers to the above questions in the context of commodity devices, *without assuming* that malware is prevented from

[1] For example, a secure key update in response to side-channel attacks – as prescribed by leakage-resilient cryptography – could not be performed with significant assurance, even if a host could reach a remote device using secure network communication channels, now assumed to be beyond an adversary's control. Device malware could always respond correctly to key update commands using the already captured device key.

accessing secrets stored on commodity devices and communicating with an external network adversary which controls it. Instead, we present a system that limits the bandwidth of the device's output channel to D_{ban} bits per second, updates secrets periodically and prevents the leakage of an entire pool of secrets. In effect, our system *confines* malware M_{in} sufficiently to enable a remote verifier established the three desirable properties outlined above, namely (1) remote-device authentication, reaching malware-free device states, and trusted (re)boot of application software. Specifically, we revisit the notion of *device attestation with bounded leakage of secrets* (DABLS) and system description provided by Tran [9], and provide new operating conditions and modes, called ReDABLS, which appear to be practical for large classes of *different* system types and configurations. In particular, we argue that in contrast to the overhead rates of DABLS, which make it impractical for large classes of intuitive operating conditions, ReDABLS can yield much lower overhead rates, particularly when acceptable probabilistic upper bounds are found for an adversary's success in attacking it.

ReDABLS also differs from the better-known software-only root of trust (SWORT) mechanisms [2, 3, 5–8] in three ways. First, SWORT mechanisms do not provide device authentication directly since they have been introduced to achieve only *authentication of program execution* on any device of the *same* type and configuration. Second, without additional mechanisms, SWORT does not usually guarantee uninterruptable composition of malware-free state establishment and trusted boot of application software; i.e., malware could re-install itself into the device after the establishment of a malware-free state and trusted software boot. Third, the ReDABLS verifier would be less susceptible to timing variations in the speed of the computing device (e.g., processor speed), since it would have to tolerate larger (e.g., network) delays, by design. For these reasons, practical answers to the three questions posed for ReDABLS above would provide stronger, more robust guarantees than SWORT.

We envision the use of ReDABLS for several applications that require periodic attestation of malware-free state and secure initialization for (1) hypervisors of remote devices, (2) control software of autonomous devices, and (3) software of unattended smart-grid devices (e.g., smart meters).

2 A Brief Overview of DABLS

In DABLS, a remote device is initialized with a device-unique, large pool of secrets S comprising N blocks of b bits each, prior to deployment;[2] viz., Fig. 1. The pool is updated from S_{i-1} to S_i by using a device local function $f(n_{i-1}, S_{i-1})$ in response to a nonce n_{i-1} sent by a remote Verifier, in every T_s seconds, where T_s represents the device computation time dedicated to application execution. The time used for updating the N blocks of the pool and responding to the Verifier is denoted by T_{up}. At the end of each update time, T_{up}, the device sends response

[2] The initialization of pool S is done using a pseudo-random number generator, PRNG, which yields statistically unique pool values for reasonable sizes of N and b.

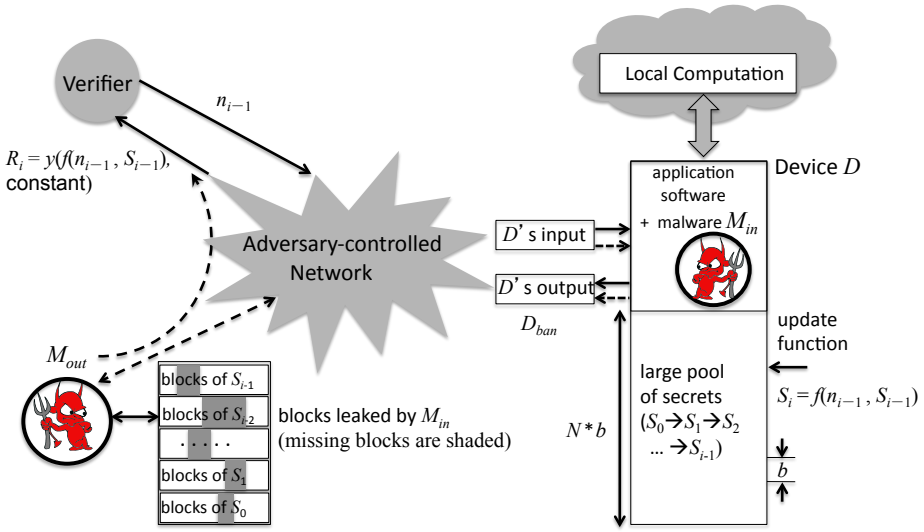

Fig. 1. A snapshot of the DABLS operation.

$R_i = y(f(n_{i-1}, S_{i-1}), constant)$ to the Verifier, where y is a message authentication code function based on the updated secret pool state $S_i = f(n_{i-1}, S_{i-1})$. Failure to respond by the end of T_{up} or to produce an incorrect response R_i causes the Verifier to signal an exception. As a result of pool initialization to state S_0, and subsequent updates, the secret pool goes through a number of device unique states $S_0, S_1, \cdots, S_{i-1}$, unless a Verifier exception interrupts this sequence and causes re-initialization. A snapshot of the DABLS operation is illustrated in Fig. 1, and the device-authentication request and response over time are illustrated in Fig. 2.

We note that pool update function can use a variety of cryptographic primitives, such as pseudo-random functions (PRFs) and one-way functions (OWFs), to ensure that entire past pools can be computed and future pools cannot be anticipated unless all pool blocks of a given state are available, except with negligible probability. Let the speed of the cryptographic primitive used by the pool update function $f(n, S)$ be C_p seconds per pool block. The *overhead rate* of the update operation is T_{up}/T_s, and the system *feasibility condition* is $T_{up}/T_s < 1$.

In this paper we argue that the overhead rate has a lower bound $N \times C_p \times D_{ban}/b$, for a large class of pool update functions. We illustrate one of the challenges posed by the design of DABLS by showing that intuitively efficient pool update functions – not just those initially considered for DABLS [9] – fail to satisfy the system feasibility condition. This motivates our introduction of *probabilistic* pool update functions and additional operating modes for ReDABLS; viz., Section 4 below.

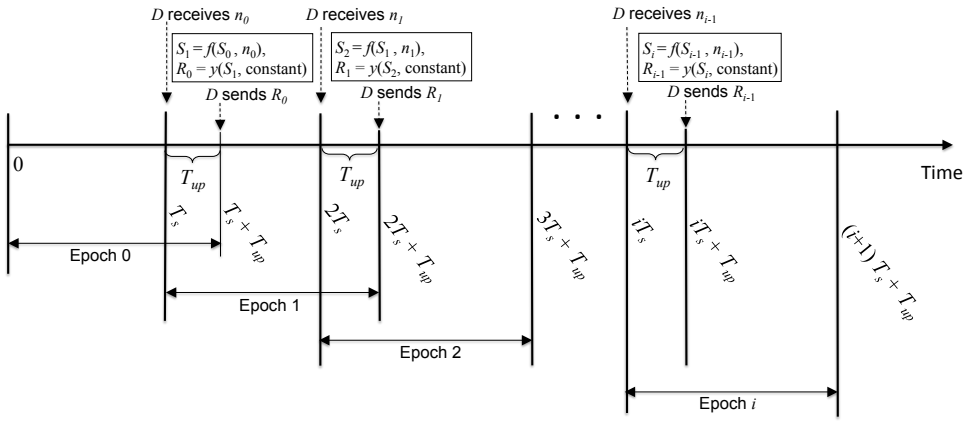

Fig. 2. The device-authentication request and response over time.

2.1 The Original Pool Update Function

DABLS presents a pool update function f_1 by using a *non-invertible* pseudo-random function (PRF) as the basic tool. Let P be a family of non-invertible PRFs. A particular non-invertible PRF P_n is selected from that family by using the nonce n; i.e., $P : \{0,1\}^{|n|} \rightarrow P_n$. An instance P_n takes as input k blocks of size b bits, and produces one block of output; namely, $P_n : \{0,1\}^{k \cdot b} \rightarrow \{0,1\}^b$.

A secret pool S is broken into N blocks of size b bits: $S[0], S[1], \ldots, S[N-1]$. $S[i]$ is referred to as block i of the pool S, where $0 \leq i \leq N-1$. As specified by Equation (1), the recursive function g takes an index i as input and produces a single block g_i as output. g_i equals $S[i]$ for $0 \leq i \leq N-1$, and is computed by inputting the N previous blocks $g_{i-j}|_{j=N,N-1,\ldots,1}$ to the PRF P_n for $i \geq N$. Finally, f_1 is realized according to Equation (2) below, where $\lambda \geq N$.

$$g_i = \begin{cases} S\left[i\right], & \text{for } 0 \leq i \leq N-1, \\ P_n(g_{i-N}||g_{i-(N-1)}|| \ldots ||g_{i-1}), & \text{for } i \geq N. \end{cases} \tag{1}$$

$$f_1(n, S) = g_\lambda||g_{\lambda+1}|| \ldots ||g_{\lambda+N-1}. \tag{2}$$

2.2 DABLS Fails the Feasibility Condition

With f_1 used as the pool update function in DABLS, a necessary condition for preventing the external adversary M_{out} from obtaining a complete secret pool is $(T_s + T_{up})D_{ban} < N \times b$, which leads to $T_s < N \times b/D_{ban}$. Let C_{P_n} be the time cost to compute P_n on each input block. Then $T_{up} = N \times \lambda \times C_{P_n}$. Therefore, the overhead rate $T_{up}/T_s > \lambda C_{P_n} D_{ban}/b \geq N C_{P_n} D_{ban}/b$. However, as illustrated in Table 1, this lower bound $N C_{P_n} D_{ban}/b$ of T_{up}/T_s is greater than 1 for typical devices, rendering DABLS infeasible in practice.

Table 1. Examples of system parameters. P_n uses AES as the block cipher and the CBC-MAC-like as the mode of encryption, as illustrated by DABLS [9].

$\|S\|$ (MB)	b (bits)	N	D_{ban} (MB/sec)	b/C_{P_n} (MB/sec)	T_{up}/T_s
0.16	128	10^4	0.001	1.2 (on ARM 1176-482MHz)	> 8.3
1.6	128	10^5	0.01	107 (on Intel Core 2-1.83 GHz)	> 9.3

Several possible approaches to reduce the lower bound $NC_{P_n}D_{ban}/b$ of T_{up}/T_s are available. For example, to decrease C_{P_n}, P_n could use a $5-8$ times faster primitive; e.g., XoR-universal hash functions. Furthermore, given a fixed $|S|$, the block size b could be set larger to decrease N. Finally, commodity hardware front ends or network interfaces that remain beyond the reach of device malware could be used to effectively decrease the outbound network bandwidth limit of the device depending on other system parameters; e.g., an increased T_s, a decreased pool size $|S|$.

3 The New Pool Update Functions for ReDABLS

DABLS fails to provide a feasible pool update function for achieving the three desirable security properties discussed in the Introduction; i.e., remote-device authentication, reaching malware-free device states, and trusted boot of application software. In ReDABLS, we rely on similar protocols as those in DABLS but introduce new pool update functions in an attempt to make the system overhead rate practical.

The challenge one faces in designing new pool update functions is that intuitive optimizations do not necessarily work. For example, one could attempt to use smaller input sizes than that of $f_1(n, S)$ without compromising update security; i.e., the number of input blocks could be smaller than N. However, this would not necessarily decrease the overhead rate; viz., function f_2 below. Another way would be to also use more efficient cryptographic primitives for the function implementation. Although, this could decrease the overhead rate, it would not necessarily satisfy our feasibility condition; viz., function f_3 below. For these reasons, we introduce probabilistic update functions (viz., f_4 and f_5 below), which can reduce the overhead rates of update functions such as f_2 and f_3 to feasible levels.

3.1 The Pool Update Function f_2

We assume that N is a product of two positive integers q and m; i.e., $N = qm$, where $m \geq 2$. Now we explain how $f_2(n, S)$ is computed given a nonce n and a secret pool S. S consists of N blocks: $S[0], S[1], \ldots, S[N-1]$. g_i is updated as follows. g_i equals $S[i]$ for $0 \leq i \leq N-1$, and is computed by inputting $(q+1)$ blocks $g_{i-jm}|_{j=q,q-1,\ldots,1}$ and g_{i-1} to the PRF P_n for $i \geq N$; namely,

$$g_i = \begin{cases} S[i], & \text{for } 0 \leq i \leq N-1, \\ P_n\big(g_{i-qm}||g_{i-(q-1)m}||\ldots||g_{i-m}||g_{i-1}\big), & \text{for } i \geq N. \end{cases} \quad (3)$$

Finally, similar to the design of f_1, the function f_2 is still defined as the last N blocks after computing λ blocks of g; i.e., $f_2(n, S)$ is set by

$$f_2(n, S) = g_\lambda || g_{\lambda+1} || \dots || g_{\lambda+N-1}. \quad (4)$$

With function f_2 used as the pool update function, we have derived a set of conditions which are both necessary and sufficient for preventing the external adversary from obtaining a complete secret pool. (A detailed analysis of these conditions is provided in Appendix A.) The set of conditions indicates that the overhead rate T_{up}/T_s is at least $NC_{P_n}D_{ban}/b$, the same lower bound as in the case where f_1 is used as the pool update function. Therefore, a system implemented with f_2 would also fail the feasibility condition $T_{up}/T_s < 1$. Note that function f_1 can be regarded as a special case of function f_2 with $q = N$.

3.2 The Pool Update Function f_3

Consider a pool update from a secret pool S to the next secret pool $f_3(n, S)$. S consists of N blocks: $S[0], S[1], \dots, S[N-1]$; $f_3(n, S)$ consists of N blocks: $y_0, y_1, \dots, y_{N-1}$. For $i = 0, 1, \dots, N-1$, block y_i is computed by inputing to PRF P_n w blocks with a sliding window of length w; specifically,

$$y_i = P_n(z_{j+i} || z_{j+i+1} || \dots || z_{j+i+(w-1)}),$$

where $j \in \{0, 1, \dots, N-1\}$ is determined by the nonce n and block z_t is defined by

$$z_t = \begin{cases} S[t], & \text{for } 0 \le t \le N-1, \\ y_{t-N}, & \text{for } t \ge N. \end{cases}$$

Here we set P_n as the MD5 hash function, whose computation cost on an input block is lower than that of CBC-MAC-AES.

With f_3 used as the pool update function, we have also proved that a necessary condition for preventing the external adversary from obtaining a complete secret pool is $T_{up}/T_s \ge NC_{P_n}D_{ban}/b$. (The proof is omitted due to space limitations.) If P_n is implemented with the MD5 function, f_3 also fails the feasibility condition $T_{up}/T_s < 1$, as illustrated in Table 2.

Table 2. Examples of system parameters. P_n is the MD5 hash function.

$\|S\|$ (MB)	b (bits)	N	D_{ban} (MB/sec)	b/C_{P_n} (MB/sec)	T_{up}/T_s
0.16	128	10^4	0.001	2 (on ARM 1176-482MHz)	> 5
1.6	128	10^5	0.01	268 (on Intel Core 2-1.83 GHz)	> 3.7

3.3 The Probabilistic Pool Update Functions f_4 and f_5

We consider probabilistic pool update functions and show that they can reduce the overhead rates of the three non-probabilistic functions discussed above to feasible levels.

Random Permutation. The idea of using a random permutation is that, given a random nonce, we can derive a permutation from the nonce that will change the ordering of the N blocks of the old secret pool randomly before the pool update. The intuition behind the overhead rate T_{up}/T_s reduction is that the external adversary M_{out} still cannot obtain a complete secret pool even though the internal adversary M_{in} leaks more blocks per epoch than in the three cases above where a random permutation is not used. This is the case because M_{in} can no longer be sure which pool blocks will be useful as the input for the update function in future epochs. Instead, the external adversary M_{in} has to predict the usefulness of blocks in future epochs.

We introduce a pool update function f_4, which is the random permutation version of function f_2, as shown below. Given a secret pool S, we now explain the computation of $f_4(n, S)$. S consists of N blocks: $S[0], S[1], \ldots, S[N-1]$. Then, based on the external nonce n, blocks $S[0], S[1], \ldots, S[N-1]$ are randomly permuted to blocks $S'[0], S'[1], \ldots, S'[N-1]$, which are inputs to the recursive block computation. Similar to (3), we define g_i as follows:

$$g_i = \begin{cases} S'[i], & \text{for } 0 \le i \le N-1, \\ P_n\big(g_{i-qm}||g_{i-(q-1)m}||\ldots||g_{i-m}||g_{i-1}\big), & \text{for } i \ge N. \end{cases}$$

Then the same as (4), $f_4(n, S)$ is set by

$$f_4(n, S) = g_\lambda||g_{\lambda+1}||\ldots||g_{\lambda+N-1}.$$

With f_4 used as the pool update function in ReDABLS, one can prove that the overhead rate is reduced by a constant factor that depends on system architecture (e.g., the ratio of the local memory size over the pool size $|S|$), compared with the case where f_2 is the pool update function.

Partial Pool Update. Consider a pool update from secret pool S_i to S_{i+1}. A fixed number of blocks in S_i are propagated to S_{i+1} without any change, while the remaining blocks in S_i are updated. The purpose of partial pool update is to reduce the number of blocks in each pool update to be updated by P_n, and thus to reduce T_{up}, which helps decrease the overhead rate T_{up}/T_s. Also, we need to enforce that the external adversary M_{out} still can obtain a complete secret pool with only a negligible probability.

We introduce a pool update function f_5, which is the partial pool update version of function f_3, as detailed below. In computing $f_5(n, S)$, let the number of propagated blocks in S be G. The G blocks are randomly selected based on

the nonce n and are uniformly distributed among the N blocks of S. Without knowing the nonce n, the attacker cannot predict the G blocks.

The maximal number of blocks leaked or saved blocks in any epoch should be less than the sliding window w; otherwise, M_{out} could splice these leaked or saved blocks and update them, obtaining at least w blocks of secret pool S_0 at the end of epoch 0, at least $(w+1)$ blocks of secret pool S_1 at the end of epoch 1, ..., and a complete secret pool S_i at the end of epoch $(N-w)$. We denote that number by $(w-h)$ since it is less than w, where $h \geq 1$. Clearly, $(w-h)$ is at least $T_s D_{ban}/b$, the maximal number of blocks leaked within T_s. Then $T_s \leq (w-h)b/D_{ban}$. Noting that $T_{up} = (N-G) \cdot w \cdot C_{P_n}$, we finally obtain

$$T_{up}/T_s \geq \frac{1-G/N}{1-h/w} \times NC_{P_n} D_{ban}/b.$$

The goal is to let the probabilistic-update factor $\frac{1-G/N}{1-h/w}$ be small and have a negligible probability for M_{out} succeeding in obtaining a complete secret pool. First, to ensure $\frac{1-G/N}{1-h/w} \leq 1$, it follows that $G \geq h$.

Given the leaked or saved $(w-h)$ blocks in an epoch and the G propagated blocks, if there are at least w consecutive blocks among them, then M_{out} succeeds in splicing the w blocks. Consider that the $(w-h)$ blocks have i (resp., j) number of the G propagated blocks to their immediate left (resp., right), where $i, j \geq 0$. M_{out} succeeds in splicing if $i + j \geq h$.

We have

$$\Pr[(i \geq h) \cap (j \geq 0)] = \binom{N-h}{G-h} \Big/ \binom{N}{G},$$

and for $t = h-1, h-2, \ldots, 0$,

$$\Pr[(i \geq t) \cap (j \geq h-t)] = \binom{N-h-1}{G-h} \Big/ \binom{N}{G}.$$

Then

$$\begin{aligned}
&\Pr[M_{out} \text{ succeeds in obtaining a complete secret pool}] \\
&\leq \Pr[i+j \geq h] \\
&= \binom{N-h}{G-h} \Big/ \binom{N}{G} + h \cdot \binom{N-h-1}{G-h} \Big/ \binom{N}{G} \\
&\leq (h+1) \times (G/N)^h.
\end{aligned}$$

Table 3 presents examples of system parameters.

Table 3. Examples of system parameters. The units of $|S|$, b, D_{ban} and b/C_{P_n} are MB, MB/sec, bits and MB/sec, respectively. P_n is the MD5 hash function; and $\Pr^*$ is $\Pr[M_{out}$ succeeds in obtaining a complete secret pool].

$\|S\|$	b	N	D_{ban}	b/C_{P_n}	G	w	h	T_{up}/T_s	$\Pr^*$
0.16	128	10^4	0.001	2 (on ARM 1176-482MHz)	$9N/10$	$10N/11$	$N/11$	> 0.56	2.3×10^{-39}
1.6	128	10^5	0.01	268 (on Intel Core 2-1.83 GHz)	$29N/30$	$30N/31$	$N/31$	> 0.13	1.0×10^{-45}

4 Additional Operational Modes for ReDABLS

In this section, we briefly outline two additional operational modes intended to enhance the usability of ReDABLS.

First, a commodity network-interface device that is beyond the reach of malware M_{in} can be connected to the remote application device to limit the malware's output bandwidth independent of any length of T_s. All other hardware communication channels of the application device are disabled so that the network-interface device provides the only Internet connection service for the remote application device. We denote by D'_{ban} the effective outbound network bandwidth limit of the device enforced by the interface device. D'_{ban} can be set to be much smaller than D_{ban}, the device D's maximum outbound network bandwidth in the absence of the interface device. This means that T_s can be increased and/or $|S|$ decreased more than before to ensure a small overhead rate.

Second, ReDABLS can have an infrequent-activation mode based on a separate private but slow/expensive channel between the verifier and a remote human operator who would be located in the vicinity of the device and could visually identify it. The verifier-operator channel is secure, since it connects the verifier to an operator's phone via an encrypted cellular network connection, which remains beyond the reach of the network adversary M_{out}. That is, we assume that the network adversary M_{out} does not control the cellular communications, and the device owner's phone or any potential malware on it. For example, this operational mode could consist of the following specific steps.

1) The verifier sends a short secret seed over the secure channel to the device owner's phone. Note that this channel would be expensive to use for the direct transfer of the entire secret pool S from the verifier to the remote device. Instead, the secret seed is used to generate the pool on the device owner's phone.
2) The device owner's phone generates a secret pool S by seeding a PRNG with the short secret, and sends S to the device via a fast private channel (e.g., through a USB cable connecting the phone to the device).
3) The verifier can now authenticate the remote device, enable a multi-epoch establishment of malware-free state, and perform a trusted boot.

Note that this operational mode would be used only infrequently. Thus even if its overhead rate would be high, the overall overhead would be acceptable because it would be chargeable only to a small portion of the device operation. This operational mode also offers an opportunity to optimize $|S|$, the size of pool S. On the one hand, $|S|$ cannot be very large since S is generated on a phone with limited battery power, and on the other, $|S|$ cannot be very small because, otherwise, secret pool S could be leaked to the network adversary M_{out} by malware M_{in} before it gets updated.

5 Conclusions

In earlier work, Tran [9] presented DABLS – a system that attempts to achieve remote-device authentication, reaching malware-free device states, and trusted boot of application software, in the continuous presence of device malware that can access device secrets and leak them to a network adversary. However, DABLS was infeasible in practice, leaving us with the substantial challenge of designing new operating conditions and modes for bounding the leakage of device secrets in a practical manner. To this end, we introduced a set of *probabilistic* update functions that decrease the update overhead rates in ReDABLS, and outlined new modes of operation to further decrease the relative system overhead. We now believe that the feasibility of device attestation with bounded leakage of secrets can be established conclusively.

Acknowledgments

The first author was supported in part by Lockheed Martin's CyLab Corporate Partners membership funds.

References

1. B. Gassend, D. Clarke, M. van Dijk, and S. Devadas. Silicon physical random functions. In *Proceedings of ACM Conference on Computer and Communication Security (CCS)*, pages 148–160, 2002.
2. X. Kovah, C.Kallenberg, C. Weathers, A. Herzog, M. Albin, and J. Butterworth. New results for timing-based attestation. In *Proceedings of the IEEE Symposium on Security and Privacy*, May 2012.
3. Y. Li, J. M. McCune, and A. Perrig. SBAP: Software-based attestation for peripherals. In *International Conference on Trust and Trustworthy Computing (TRUST)*, June 2010.
4. R. Pappu. *Physical One-Way Functions*. PhD thesis, MIT School of Architecture and Planning, Program in Media Arts and Sciences, Mar. 2001.
5. A. Seshadri, M. Luk, A. Perrig, L. van Doorn, and P. Khosla. SCUBA: Secure code update by attestation in sensor networks. In *Proceedings of ACM Workshop on Wireless Security (WiSe)*, Sept. 2006.
6. A. Seshadri, M. Luk, E. Shi, A. Perrig, L. van Doorn, and P. Khosla. Pioneer: Verifying integrity and guaranteeing execution of code on legacy platforms. In *Proceedings of ACM Symposium on Operating Systems Principles (SOSP)*, pages 1–16, Oct. 2005.

7. A. Seshadri, A. Perrig, L. van Doorn, and P. Khosla. SWATT: Software-based attestation for embedded devices. In *Proceedings of the IEEE Symposium on Security and Privacy*, May 2004.
8. M. Shaneck, K. Mahadevan, V. Kher, and Y. Kim. Remote software-based attestation for wireless sensors. In *Proceedings of ESAS*, 2005.
9. A. Tran. DABLS: Device attestation with bounded leakage of secrets. Master's thesis, Carnegie Mellon University, July 2011. Available online at http://www.cylab.cmu.edu/files/pdfs/tech_reports/CMUCyLab13009.pdf.
10. Trusted Computing Group. Trusted platform module main specification, Part 1: Design principles, Part 2: TPM structures, Part 3: Commands. Version 1.2, Revision 103, July 2007.

A Analysis of ReDABLS with Pool Update Function f_2

In this appendix, we provide a detailed analysis of ReDABLS only for the pool update function f_2.[3]

A.1 Minimum Amount of Leakage Necessary

In this update function, we enforce the condition

$$\lambda \geq N + m - 1 \tag{5}$$

to ensure that any block of the updated pool $f(n, S)$ ultimately needs all N blocks of previous secret pool S as inputs. If this condition is satisfied, then the internal adversary (i.e., malware) M_{in} needs to leak *at least* $N = m \cdot q$ blocks to succeed in leaking an entire secret pool.

Let S be the concatenation of N blocks $x_{D+i}|_{i=0,1,\ldots,N-1}$, where $D \geq 0$. For $j = 0, 1, \ldots, m-1$, we define $Y_j := \{x_{D+j+\ell m}\}|_{\ell=0,1,\ldots,m-1}$. Clearly, the computation of block x_{D+N+j} needs $x_{D+N+j-1}$ and all blocks in Y_j. Therefore, by a recursive analysis, the condition $j \geq m-1$ is necessary and sufficient to ensure that block x_{D+N+j} ultimately needs all blocks in $\bigcup_{j=0}^{m-1} Y_j$.

We denote the initial secret pool by S_0 and denote the N blocks of S_0 by $x_0, x_1, \ldots, x_{N-1}$. Let the secret pool obtained in the ith pool update be S_i. In the ith pool update for $i \geq 1$, pool S_{i-1} is updated to S_i. We let A_i be the set of blocks computed by the update. Then for $i \geq 1$, set A_i consists of all λ blocks with indices at least $(i-1)\lambda + N$ and at most $i\lambda + N - 1$; i.e., $A_i = \{x_{(i-1)\lambda+N}|_{j=0,1,\ldots,\lambda-1.}\}$. We also define A_0 as the set of N blocks $x_0, x_1, \ldots, x_{N-1}$; i.e., $A_0 = \{x_j|_{j=0,1,\ldots,N-1.}\}$. Note that blocks in A_i can only be leaked after time iT_s for $i \geq 0$.

A.2 Preventing the Leakage of N Contiguous Blocks

Another necessary condition for attack success is for the external adversary M_{out} to obtain an arbitrary group of N *contiguous* blocks among the blocks leaked

[3] For simplicity, we sometimes drop the subscript of f_2 and just use f to denote f_2.

by malware M_{in}. (Note that we do not consider brute-force attacks whereby the adversary attempts to discover a few pool bits that are not available in the leaked blocks). We call a leaked block x "useful" in satisfying the contiguity condition, if x is used in at least one intermediate computation to obtain one of the final N desired blocks or if x itself is one of the final N desired blocks. Without loss of generality, we only consider attacks in which all the leaked blocks are useful. The *index stretch* of an attack is defined as the result of the highest index of useful leaked blocks minus the lowest index of useful leaked blocks.

For the external adversary M_{out} to obtain the final N consecutive blocks, the internal adversary M_{in} has to leak at least N blocks. For any successful attack with a certain number of useful leaked blocks, we derive the maximal index stretch of the attack to see how M_{in} might be able to spread the leaked blocks in multiple epochs so that only a small number of blocks needs to be leaked in each epoch. To this end, we establish Theorem 1. Preventing the maximal index stretch will deny M_{in} the opportunity to leak useful blocks.

Theorem 1 *The maximal index stretch of a successful attack with a fixed number M of useful leaked blocks is $(M-N)m+qm^2-2m+1$, where $M \geq N$.*

We use the following Lemmas 1-4 in the proof of Theorem 1. The proofs of Theorem 1, and Lemmas 1-4 are given in Section A.5 of the appendix.

Lemma 1 *The maximal index stretch of a successful attack with N useful leaked blocks is qm^2-2m+1.*

Lemma 2 *For the external adversary M_{out} to obtain J blocks out of N consecutive blocks $x_{G+i}|_{i=0,1,\ldots,N-1}$, where $G \geq 0$ and $1 \leq J \leq N$, the internal adversary M_{in} has to leak at least J blocks with indices at most $G+N-1$.*

Lemma 3 *Consider the following attack with M leaked blocks, where $M \geq N$. With $y := (M-N) \bmod q$ and $a := M-N-qm$, the internal adversary M_{in} leaks the following M blocks:*

$$x_{A-ym-iqm+jm}|_{\substack{i=0,1,\ldots,a;\\ j=0,1,\ldots,q-1}}, x_{A+im}|_{i=q-y+1,\ldots,q-1},$$
$$x_{A-ym-iqm+qm-1}|_{i=0,1,\ldots,a}, x_{A+iqm-i+jm}|_{\substack{i=1,\ldots,m-1;\\ j=0,1,\ldots,q-1.}}$$

Then the external adversary M_{out} can obtain any block with index at least $qm^2-qm-2m+2$; and the index stretch of this attack is $(M-N)m+qm^2-2m+1$.

Lemma 4 *If adversary M_{out} obtains blocks $x_{A+iqm-i-(\sum_{\ell=1}^{i}\beta_\ell)m+jm}|_{j=0,1,\ldots,q-1}$ and $x_{A+(i+1)qm-(i+1)-(\sum_{\ell=1}^{i}\beta_\ell)m}$ for $i=0,1,\ldots,r$, where $r \geq 1$, then adversary M_{out} can acquire blocks in*

$$\begin{aligned}&\mathcal{B}_i :\\ &= \{x_{A+iqm-i-\left(\sum_{\ell=1}^{i}\beta_\ell\right)m+zm}|_{z=0,1,\ldots,(r+1)q-iq-\sum_{\ell=i+1}^{r}\beta_\ell}\},\end{aligned} \tag{6}$$

for $i=0,1,\ldots,r$.

A.3 Non-circumventable Time-Space Tradeoff

Internal adversary (i.e., malware) M_{in} faces a space-time tradeoff in its attempts to leak N contiguous pool blocks to M_{out}, whenever the output bandwidth D_{ban} prohibits the transfer of all N block in a single epoch. This is the case because M_{in} would either have to save the blocks not leaked in an epoch in freely usable system memory, denoted by L_{mem} below, for leakage in future epochs or perform the pool update computation using fewer than N blocks, or both. Hence, either (1) the memory size of L_{mem} is large enough to hold most, if not all, the blocks not leaked in an epoch or (2) M_{in} would have to use extra update computation time, if not enough memory space is available in L_{mem}. In the latter case, some of the N pool blocks would have to remain unused during the update, and thus extra computation time would be needed. However, if both L_{mem} and the pool update time T_{up} are upper-bounded by appropriately small values, M_{in} will either not have enough memory space to leak all blocks or will exceed the update time and be detected by the verifier. We call this the *space-time tradeoff* faced by M_{in}. If M_{in} cannot circumvent this tradeoff, it could not leak all the contiguous N blocks of any complete secret pool S.

Upper bound on memory freely usable by malware. We derive the upper bound on L_{mem}, the amount of memory freely usable by malware M_{in}, so that it cannot circumvent the time-space tradeoff.

From Theorem 1, for any successful attack with $(N+y)$ leaked blocks, where $y \geq 0$, its index stretch is no greater than $(qm^2 - 2m + 1 + ym)$. Recall that $A_i = \{x_{(i-1)\lambda+N+j}|_{j=0,1,\ldots,\lambda-1.}\}$ and $|A_i| = \lambda$ for $i \geq 1$, $A_0 = \{x_j|_{j=0,1,\ldots,N-1.}\}$ and $|A_0| = N$. Therefore, the $(N + y)$ blocks leaked by M_{in} fall in at most $\left(\left\lfloor \frac{qm^2-2m+ym}{\lambda} \right\rfloor\right) + 2$ (denoted by L hereafter) number of successive sets among $A_i|_{i=0,1,\ldots.}$. Assume the $(N+y)$ leaked blocks fall in h successive sets A_H, A_{H+1}, $\ldots, A_{H+h-1}$, where $1 \leq h \leq L$. Let F_i be the set of bits which are leaked by M_{in} and are among the bits in blocks of A_i, for $i = H, H+1, \ldots, H+h-1$. Recall that blocks in A_i can only be leaked after time iT_s for $i \geq 0$. We actually give adversary M_{in} more power by assuming that M_{in} can leak blocks in A_i at any time instance immediately after time iT_s, since the computations of blocks in A_i start from time iT_s and finish before $iT_s + T_{up}$.

After time HT_s, adversary M_{in} can leak bits in F_H. At time $(H+1)T_s$, the number of bits in F_H that M_{in} has not leaked is at least $\max\{|F_H| - T_s D_{ban}, 0\} \geq |F_H| - T_s D_{ban}$. After time $(H+1)T_s$, adversary M_{in} can leak bits in F_{H+1} or leak those bits in F_H that has not been leaked. At time $(H+2)T_s$, the bits in $F_H \cup F_{H+1}$ that M_{in} has not leaked is at least $\max\{\max\{|F_H| - T_s D_{ban}, 0\} + |F_{H+1}| - T_s D_{ban}, 0\} \geq |F_H| + |F_{H+1}| - 2T_s D_{ban}$.

This process continues iteratively. Then, at time $(H+h)T_s$, the bits in $\bigcup F_{H+j}|_{j=0,1,\ldots,h-1}$ that M_{in} has not leaked is at least $\sum_{j=0,1,\ldots,h-1} |F_{H+j}| - hT_s D_{ban}$. From time $(H+h)T_s$ to $(H+h)T_s + T_{up}$, the pool S_{H+h-1} is updated to S_{H+h}; and the set of computed blocks is A_{H+h}. At time $(H+h)T_s + T_{up}$, the bits in $\bigcup F_{H+j}|_{j=0,1,\ldots,h-1}$ that M_{in} has not leaked is at least $\sum_{j=0,1,\ldots,h-1} |F_{H+j}| -$

$(hT_s + T_{up})D_{ban}$. If all blocks of S_{H+h} are in the memory, L_{mem} is the available space to store the bits in $\bigcup F_{H+j}|_{j=0,1,\ldots,h-1}$ that has not been leaked. Therefore, to ensure that malware M_{in} cannot circumvent the space-time tradeoff and leak all bits in $\bigcup F_{H+j}|_{j=0,1,\ldots,h-1}$, we impose the condition

$$\sum_{j=0}^{h-1} |F_{H+j}| - (hT_s + T_{up})D_{ban} > L_{mem} \tag{7}$$

Given $\sum_{j=0}^{h-1} |F_{H+j}| = (N+y)b$ and $1 \leq h \leq L$, where $L = \left(\left\lfloor \frac{qm^2-2m+ym}{\lambda} \right\rfloor\right) + 2$, then we obtain

$$\begin{aligned} &\text{L.H.S. of (7)} \\ &\geq (N+y)b - \left[\left(\frac{qm^2-2m+ym}{\lambda} + 2\right) T_s + T_{up}\right] D_{ban} \end{aligned} \tag{8}$$

R.H.S. of (8) increases as y increases if

$$b\lambda \geq mT_s D_{ban}. \tag{9}$$

Therefore, we enforce (9) and

$$N \cdot b - \left[\left(\frac{qm^2-2m}{\lambda} + 2\right) T_s + T_{up}\right] D_{ban} > L_{mem}, \tag{10}$$

so that (7) follows for any $y \geq 0$ and $h = 1, 2, \ldots, L$.

Computation Cost of Pool Update. Using the terminology of DABLS [9], we call the case when all $|S|$ bits of memory are used for the computation of $f(n, S)$, the *benign case*, and the case when fewer than $|S|$ bits of memory are used for computation, the *malicious case*. The following theorem gives the computation cost of $f(n, S)$ in both the benign and malicious cases.

Theorem 2 *For a pool update $f(n, S)$ in the benign case, the computation cost is $\lambda(q+1)C_{P_n}$. For a pool update $f(n, S)$ in the malicious case, if $\lambda \geq N + m - 1$ and c blocks of memoization cache [9] are used, where $c < N$, then the computation cost is at least $[2^{\lceil \frac{\lambda-1-c}{m} \rceil - 2} m(m+1)(q+1) + c(q+1)]C_{P_n}$.*

The proof of Theorem 2 is given in Section A.5 of this appendix.

Remark 1 *If q is a constant and does not scale with N, then the computation cost in the benign case is linear with λ.*

When $|S|-1$ bits are used, this leaves $c = N-1$ blocks to cache intermediate blocks. Given $c = N-1$ and Theorem 2, we enforce the relation:

$$\lambda(q+1)C_{P_n} < T_{up} < [2^{\lceil \frac{\lambda-N}{m} \rceil - 2} m(m+1)(q+1) + (N-1)(q+1)]C_{P_n}. \tag{11}$$

A.4 Summary of ReDABLS Parameter Conditions for $f_2(n, S)$

From the above analysis, the required parameter conditions are (5), (9), (10), and (11), whenever $N = m \cdot q$ and $m \geq 2$.

A.5 Proof of Theorems 1, 2 and Lemmas 1-4

Proof of Theorem 1. Among the successful attacks with a fixed number M of useful leaked blocks, where $M \geq N$, let $\mathcal{A}_M$ be an attack which maximizes the index stretch and I_M be the index stretch of $\mathcal{A}_M$. As explained in Section A.2, all leaked blocks in $\mathcal{A}_M$ are "useful." We regard $M \geq N + 1$ below.

Let $x_{B+j}|_{j=0,1,\ldots,N-1}$ be the N consecutive blocks that the external adversary M_{out} finally obtains. Note that M_{out} can further use $x_{B+j}|_{j=0,1,\ldots,N-1}$ to get any block with an index of at least B. Denote the useful block with the lowest index in attack $\mathcal{A}_M$ by x_A. To make x_A useful, the external adversary M_{out} should obtain $x_{A+jm}|_{j=0,1,\ldots,q-1}, x_{A+qm-1}$ to compute x_{A+qm}. Since x_A is the useful block with the lowest index, all of $x_{A+jm}|_{j=0,1,\ldots,q-1}, x_{A+qm-1}$ can only be leaked instead of being computed, given the fact that if at least one of $x_{A+jm}|_{j=1,2,\ldots,q-1}, x_{A+qm-1}$ is computed, then at least one block with an index lower than A should be leaked. Clearly, x_A is not used in computation(s) other than that of x_{A+qm}.

We note that the highest index among the leaked blocks is $B+N-1$ since any block with index greater than $B+N-1$ is useless in inducing $x_{B+j}|_{j=0,1,\ldots,N-1}$. Then the index stretch I_M is at most $B + N - 1 - A$. Lemma 3 presents a successful attack with a fixed number M of useful leaked blocks and with an index stretch of $qm^2 - 2m + 1 + (M - N)m$. Hence, it follows that

$$B + N - 1 - A \geq I_M \geq qm^2 - 2m + 1 + (M - N)m. \tag{12}$$

Given $M \geq N + 1$, $N = qm$ and (12), it holds that for $i = 1, 2, \ldots, m - 1$,

$$B > A + iqm - i. \tag{13}$$

Consequently, block $x_{A+iqm-i}$ is not one of the final desired blocks $x_{B+j}|_{j=0,1,\ldots,N-1}$ for each $i = 1, 2, \ldots, m - 1$.

We have the following observation. If x_{A+qm-1} is used in computations in addition to that of x_{A+qm}, then there exists $\beta_1 \in \{0, 1, \ldots, q-1\}$ such that adversary M_{out} obtains $x_{A+qm-1-\beta_1 m+jm}|_{j=0,1,\ldots,q-1}$ and $x_{A+2qm-2-\beta_1 m}$, which are together used to compute $x_{A+2qm-1-\beta_1 m}$. If $x_{A+2qm-2-\beta_1 m}$ is used in computations in addition to that of $x_{A+2qm-1-\beta_1 m}$, then there exists $\beta_2 \in \{0, 1, \ldots, q - 1\}$ such that adversary M_{out} obtains $x_{A+2qm-2-(\beta_1+\beta_2)m+jm}|_{j=0,1,\ldots,q-1}$ and $x_{A+3qm-3-(\beta_1+\beta_2)m}$, which are together used to compute $x_{A+3qm-2-(\beta_1+\beta_2)m}$.

This process continues iteratively. Then we have the following two cases. 1) There exist $\beta_\ell \in \{0, 1, \ldots, q-1\}$ for $\ell = 1, 2, \ldots, m-1$ such that $x_{A+iqm-i-(\sum_{\ell=1}^{i-1} \beta_\ell)m}$ is used in computations in addition to that of $x_{A+iqm-(i-1)}$ for $i = 1, 2, \ldots, m-1$. 2) There exist $\gamma \in \{1, 2, \ldots, m-1\}$ and $\beta_\ell \in \{0, 1, \ldots, q-1\}$ for $\ell = 1, 2, \ldots, \gamma-1$ such that $x_{A+iqm-i-(\sum_{\ell=1}^{i-1} \beta_\ell)m}$ is used in computations in addition to that of

$x_{A+iqm-(i-1)-(\sum_{\ell=1}^{i-1}\beta_\ell)m}$ for $i = 1, 2, \ldots, \gamma-1$; and $x_{A+\gamma qm-\gamma-(\sum_{\ell=1}^{\gamma-1}\beta_\ell)m}$ is not used in any computation other than that of $x_{A+\gamma qm-(\gamma-1)-(\sum_{\ell=1}^{\gamma-1}\beta_\ell)m}$.

We first consider case 1). Defining $\sum_{\ell=1}^{i}\beta_\ell = 0$ for $i = 0$, then by an iterative analysis, we know that for $i = 0, 1, \ldots, m-1$, adversary M_{out} has $x_{A+iqm-i-(\sum_{\ell=1}^{i}\beta_\ell)m+jm}|_{j=0,1,\ldots,q-1.}$ and $x_{A+(i+1)qm-(i+1)-\left(\sum_{\ell=1}^{i}\beta_\ell\right)m}$, and use them to compute $x_{A+(i+1)qm-i-\left(\sum_{\ell=1}^{i}\beta_\ell\right)m}$. From Lemma 4, adversary M_{out} further acquires all blocks in $\cup_{i=0}^{m-1}\mathcal{B}_i$, where $\mathcal{B}_i$ is defined by (6). Define

$$T := A + (m-1)qm - (m-1) - \left(\sum_{\ell=1}^{m-1}\beta_\ell\right)m \tag{14}$$

Then given $\beta_\ell \in \{0, 1, \ldots, q-1\}$ for $\ell = 1, 2, \ldots, m-1$, the lowest index among the blocks in $\mathcal{B}_i$ is $A+iqm-i-\left(\sum_{\ell=1}^{i}\beta_\ell\right)m$, which is at least $T+(m-i-1)$; and the highest index among the blocks in $\mathcal{B}_i$ is $A+mqm-i-\left(\sum_{\ell=1}^{m-1}\beta_\ell\right)m$, which equals $T+qm+(m-i-1)$. Therefore, for $i = 0, 1, \ldots, m-1$, adversary M_{out} obtains at least all the blocks whose indices modulo m give $(T-i-1) \bmod m$ and whose indices are at least $T+(m-i-1)$ and at most $T+qm+(m-i-1)$. In other words, adversary M_{out} obtains at least all the blocks whose indices are at least T and at most $T+qm+m-1$. Clearly, M_{out} acquires blocks $x_{T+j}|_{j=0,1,\ldots,qm-1}$ without leaking without leaking x_i for any $i > T+qm-1$. Given obtained $x_{T+j}|_{j=0,1,\ldots,qm-1}$, M_{out} can further compute x_i for any $i > T+qm-1$. Then M_{out} obtains any block with index at least T. From (13), $B > T$ follows. Hence, M_{out} gets the final desired N blocks $x_{B+j}|_{j=0,1,\ldots,N-1}$ without leaking x_i for any $i > T+qm-1$. Then it further follows that I_M (i.e., the index stretch of attack $\mathcal{A}_M$) is at most $T+qm-1-A$, which is at most qm^2-m given the definition of T in (14) and $\beta_\ell \geq 0$ for $\ell = 1, 2, \ldots, m-1$. This contradicts with (12) which shows that I_M is at least $qm^2-2m+1+(M-N)m$ and thus at least qm^2-m+1, given $M \geq N+1$. Hence, case 1) does not hold.

Then we consider case 2). Noting $\sum_{\ell=1}^{i}\beta_\ell = 0$ for $i = 0$, we know that for $i = 0, 1, \ldots, \gamma-1$, adversary M_{out} has $x_{A+iqm-i-(\sum_{\ell=1}^{i}\beta_\ell)m+jm}|_{j=0,1,\ldots,q-1.}$ and $x_{A+(i+1)qm-(i+1)-\left(\sum_{\ell=1}^{i}\beta_\ell\right)m}$, and use them to compute $x_{A+(i+1)qm-i-\left(\sum_{\ell=1}^{i}\beta_\ell\right)m}$. Note that given $\beta_i \in \{0, 1, \ldots, q-1\}$, block $x_{A+(i+1)qm-(i+1)-\left(\sum_{\ell=1}^{i}\beta_\ell\right)m}$ belongs to $x_{A+(i+1)qm-(i+1)-(\sum_{\ell=1}^{i+1}\beta_\ell)m+jm}|_{j=0,1,\ldots,q-1.}$. We define T as the N consecutive blocks with indices starting from $A+(\gamma-1)qm+1-(\sum_{\ell=1}^{\gamma-1}\beta_\ell)m$ and ending with $A+\gamma qm-(\sum_{\ell=1}^{\gamma-1}\beta_\ell)m$. Let T_1 be the set of blocks which belong to T and whose indices modulo m give $(A-i) \bmod m$ for $i = 0, 1, \ldots, \gamma-1$. Then $|T_1| = \gamma q$. From Lemma 4, adversary M_{out} obtains at least all blocks in T_1. Other than $x_{A+\gamma qm-\gamma-(\sum_{\ell=1}^{\gamma-1}\beta_\ell)m}$ and the blocks in T_1, among the blocks in T, let T_2 be the set of remaining blocks that adversary M_{out} obtains. Then among the blocks in T, adversary M_{out} obtains $T_* := T_1 \cup T_2 \cup \{x_{A+\gamma qm-\gamma-(\sum_{\ell=1}^{\gamma-1}\beta_\ell)m}\}$. From Lemma 2, to acquire T_*, adversary M_{in} has to leak at least T_* blocks with indices at most $A+\gamma qm-(\sum_{\ell=1}^{\gamma-1}\beta_\ell)m$.

We refer attack A_M as A_M^* when M_{in} leaks exactly the following $T_* = (\gamma q + |T_2| + 1)$ blocks among the blocks with indices at most $A + \gamma qm - (\sum_{\ell=1}^{\gamma-1} \beta_\ell)m$:

$$x_{A+iqm-i-(\sum_{\ell=1}^{i}\beta_\ell)m+jm}\big|_{\substack{i=0,1,\ldots,\gamma-1;\\ j=0,1,\ldots,q-1.}} \tag{15}$$

$$x_{A+\gamma qm-\gamma-(\sum_{\ell=1}^{\gamma-1}\beta_\ell)m}, \text{ and all blocks in } T_2. \tag{16}$$

Then the number of leaked blocks with indices at most $A+\gamma qm-(\sum_{\ell=1}^{\gamma-1}\beta_\ell)m$ in any instance of A_M is at least that in any instance of A_M^*. Therefore, considering that (a) the number of leaked blocks with indices greater than $A + \gamma qm - (\sum_{\ell=1}^{\gamma-1}\beta_\ell)m$ in any instance of A_M is at most that in any instance of A_M^* as the total number of leaked blocks is M for both A_M and A_M^*; and (b) for both the instance of A_M and the the instance of A_M^*, T_* is the set of all obtained blocks in the N consecutive blocks T, we know an instance of A_M^* which maximizes the index stretch is also an instance of A_M which maximizes the index stretch. Hence, we can just let A_M be A_M^* in the analysis. Accordingly, we can assume the $(|T_1| + |T_2| + 1)$ blocks given by (15) (16) are leaked in attack A_M.

Given attack $\mathcal{A}_M$, we construct attack $\mathcal{A}_{M-1}$ as follows. The $(\gamma+1)$ blocks $x_{A+iqm-i-(\sum_{\ell=1}^{i}\beta_\ell)m}|_{i=0,1,\ldots,\gamma}$ leaked in $\mathcal{A}_M$ are not leaked in $\mathcal{A}_{M-1}$. The γ blocks $x_{A+iqm-(i-1)-(\sum_{\ell=1}^{i-1}\beta_\ell)m}|_{i=1,2,\ldots,\gamma}$ not leaked in $\mathcal{A}_M$ are leaked in $\mathcal{A}_{M-1}$. Other than the $(2\gamma+1)$ blocks mentioned above, the remaining blocks leaked in $\mathcal{A}_M$ are still leaked in $\mathcal{A}_{M-1}$. Then the blocks which are leaked in $\mathcal{A}_{M-1}$ and have indices at most $A + \gamma qm - (\sum_{\ell=1}^{\gamma-1}\beta_\ell)m$ constitute the set

$$\begin{aligned}
&\{x_{A+iqm-i-(\sum_{\ell=1}^{i}\beta_\ell)m+jm}\big|_{\substack{i=0,1,\ldots,\gamma-1;\}\\ j=1,\ldots,q-1.}}\\
&\cup \{x_{A+iqm-(i-1)-(\sum_{\ell=1}^{i-1}\beta_\ell)m}|_{i=1,2,\ldots,\gamma.}\} \cup T_2\\
&= \{x_{A+iqm-i-(\sum_{\ell=1}^{i}\beta_\ell)m+jm+m}\big|_{\substack{i=0,1,\ldots,\gamma-1;\}\\ j=0,1,\ldots,q-1.}} \cup T_2.
\end{aligned} \tag{17}$$

Therefore, each block given by (15) and leaked in $\mathcal{A}_M$ is now replaced with a block with index adding m in $\mathcal{A}_{M-1}$. Similar to the proof of Lemma 4, we can show by mathematical reduction that in $\mathcal{A}_{M-1}$, with leaked blocks

$$x_{A+iqm-i-(\sum_{\ell=1}^{i}\beta_\ell)m+jm+m}\big|_{\substack{i=0,1,\ldots,\gamma-1;,\\ j=0,1,\ldots,q-1.}} \tag{18}$$

adversary M_{out} can obtain all the blocks whose indices modulo m give $(A - i)$ mod m and whose indices are at least $A + iqm - i - (\sum_{\ell=1}^{i}\beta_\ell)m + m$ and at most $A + \gamma qm - i - (\sum_{\ell=1}^{\gamma-1}\beta_\ell)m$. Then (a) in attack $\mathcal{A}_{M-1}$, among the blocks in T, adversary M_{out} obtains $T_1 \cup T_2$. It's straightforward to see that any block given by (18) is useful in inducing $T_1 \cup T_2$. Then since all the M blocks in attack $\mathcal{A}_M$ are useful, all the $(M-1)$ blocks in attack $\mathcal{A}_M$ are also useful. (b) Recall that in attack $\mathcal{A}_M$, among the blocks in T, adversary M_{out} obtains $T_1 \cup T_2 \cup \{x_{A+\gamma qm-\gamma-(\sum_{\ell=1}^{\gamma-1}\beta_\ell)m}\}$, but $x_{A+\gamma qm-\gamma-(\sum_{\ell=1}^{\gamma-1}\beta_\ell)m}$ is not used in any computation other than that of $x_{A+\gamma qm-(\gamma-1)-(\sum_{\ell=1}^{\gamma-1}\beta_\ell)m}$, which is also a block in T and is leaked in $\mathcal{A}_{M-1}$. Note that (c) in attacks $\mathcal{A}_M$ and $\mathcal{A}_{M-1}$,

the leaked blocks with indices greater than $A + \gamma qm - (\sum_{\ell=1}^{\gamma-1} \beta_\ell)m$ (i.e., the highest block index of T) are the same. Therefore, from (a) and (b) and (c), among the blocks with indices greater than $A + \gamma qm - \gamma - (\sum_{\ell=1}^{\gamma-1} \beta_\ell)m$, the blocks that M_{out} can obtain in attacks $\mathcal{A}_M$ and $\mathcal{A}_{M-1}$ are the same. From (13), the lowest block index (i.e., B) among the final desired blocks $x_{B+j}|_{j=0,1,\ldots,N-1}$ is greater than $A + \gamma qm - \gamma - (\sum_{\ell=1}^{\gamma-1} \beta_\ell)m$. Then since adversary M_{out} obtains $x_{B+j}|_{j=0,1,\ldots,N-1}$ in attack $\mathcal{A}_M$, M_{out} also obtains $x_{B+j}|_{j=0,1,\ldots,N-1}$ in attack $\mathcal{A}_{M-1}$. Because x_A is the leaked block with the lowest index in $\mathcal{A}_M$; block x_{A+m} is leaked in $\mathcal{A}_{M-1}$; and the highest block index in $\mathcal{A}_{M-1}$ is no less than that in $\mathcal{A}_M$, then the index stretch of attack $\mathcal{A}_M$ minus m. Recall that $(M-1)$ blocks are leaked in attack $\mathcal{A}_{M-1}$; and I_{M-1} is the maximal index stretch of a successful attack with $(M-1)$ useful leaked blocks. Therefore, for $M \geq N+1$, it follows that $I_{M-1} \geq I_M - m$, which together with Lemma 1 and (12), leads to $I_M = qm^2 - 2m + 1 + (M-N)m$. □

Proof of Lemma 1. Here we also let $x_{B+j}|_{j=0,1,\ldots,N-1}$ be the final N consecutive blocks that the external adversary M_{out} wants to obtain. Given $N = mq$, we divide $\{x_{B+j}|_{j=0,1,\ldots,N-1}\}$ into m sets $R_i|_{i=0,1,\ldots,m-1}$, where $R_i := \{x_{B+jm+i}|_{j=0,1,\ldots,q-1}\}$ for $i = 0, 1, \ldots, m-1$. We will prove that for adversary M_{out} to acquire $x_{B+j}|_{j=0,1,\ldots,N-1}$, the internal adversary M_{in} has to leak at least N blocks in the union of m sets $G_i|_{i=0,1,\ldots,m-1}$, with $G_i := \{x_{\alpha_i m+jm+i}|_{j=0,1,\ldots,q-1}\}$, where α_i is a positive integer, for $i = 0, 1, \ldots, m-1$. Given $i = 0, 1, 2, \ldots, m-1$, if all blocks of R_i are leaked, we just set G_i as R_i. Then we consider that at least one block (denoted by x_I hereafter) of R_i is not leaked given $i = 0, 1, 2, \ldots, m-1$, where $I \bmod m = i$. Computing block x_I needs blocks $x_{I-jm}|_{j=q,q-1,\ldots,1}, x_{j-1}$. If the q blocks $x_{I-jm}|_{j=q,q-1,\ldots,1}$ are all leaked, $x_{I-jm}|_{j=q,q-1,\ldots,1}$, then we set G_i as $x_{I-jm}|_{j=q,q-1,\ldots,1}$. If there exists a j_* such that x_{I-j_*m} is not leaked, the analysis continues iteratively as computing block x_{I-j_*m} needs blocks $x_{I-j_*m-jm}|_{j=q,q-1,\ldots,1}, x_{j-1}$. Therefore, at the end M_{in} should always leak G_i to let M_{out} get G_i.

To maximize the index stretch, we consider the m sets $G_i|_{i=0,1,\ldots,m-1}$ do not "cross" with each other. In other words, there exist distinct $i_0, i_1, \ldots, i_{m-1}$ which are from $\{0, 1, \ldots, m-1\}$ such that the highest block index of $G_{i_{j-1}}$ is less than the lowest block index of G_{i_j}, where $j = 1, 2, \ldots, m-1$. Then it's straightforward to see

$$\begin{aligned}(\alpha_{i_j} m + i_j) - [\alpha_{i_{j-1}} m + (q-1)m + i_{j-1}] \\ = m - 1, \forall j = 1, 2, \ldots, m-1,\end{aligned} \tag{19}$$

so that adversary M_{out} can use $\{x_{\alpha_{i_{j-1}} m+\ell m+i}|_{\ell=0,1,\ldots,q-1}\}$ and $x_{\alpha_{i_j} m+i_j}$ to compute $x_{\alpha_{i_j} m+i_j+1}$. Given (19), with $A := \alpha_{i_0} m + i_0$, adversary M_{out} leaks the following N blocks: $x_{A+iqm-i+jm}|_{\substack{i=0,1,\ldots,m-1;\\ j=1,2,\ldots,q-1.}}$. With $x_{A+iqm-i+jm}|_{\substack{i=0,1,\ldots,m-1;,\\ j=0,1,\ldots,q-1.}}$ adversary M_{out} can further acquire any block with index at least $qm^2 - qm - 2m + 2$. In addition, any leaked block is useful in obtaining N consecutive blocks

with indices at least $qm^2 - qm - 2m + 2$. The index stretch of this attack is

$$(m-1)qm - (m-1) + (q-1)m = qm^2 - 2m + 1.$$

□

Proof of Lemma 2. The J blocks that adversary M_{out} wants to obtain can be divided into m sets $L_i|_{i=0,1,\ldots,m-1}$, where L_i consists of blocks which are part of the J blocks and whose indices modulo m all give i, for $i = 0, 1, \ldots, m-1$. Note that $|L_i| \leq q$. We show that for M_{out} to obtain L_i, M_{in} has to leak at least $|L_i|$ blocks whose indices modulo m all give i. This is true when all blocks of L_i are leaked, and is also true in the case where at least one block (say block x) of L_i is computed instead of being leaked because in that case, at least q blocks whose indices modulo m all give i should be leaked to compute block x. Hence, to obtain J blocks out of $x_{G+i}|_{i=0,1,\ldots,N-1}$, adversary M_{in} has to leak at least J blocks with indices at most $G + N - 1$. □

Proof of Lemma 3. First, given $x_{A-ym-iqm+jm}|_{\substack{i=0,2,\ldots,a;\\ j=0,1,\ldots,q-1}}$, $x_{A-ym-iqm+qm-1}|_{i=0,2,\ldots,a}$ and $x_{A+im}|_{i=q-y+1,\ldots,q-1}$, adversary M_{out} can obtain $x_{A+jm}|_{j=0,1,\ldots,q-1.}$. Then as explained in the proof of Lemma 1, with $x_{A+iqm-i+jm}|_{\substack{i=0,1,\ldots,m-1;\\ j=0,1,\ldots,q-1.}}$, adversary M_{out} can further acquire any block with index at least $qm^2 - qm - 2m + 2$. In addition, any leaked block is useful in obtaining N consecutive blocks with indices at least $qm^2 - qm - 2m + 2$. It's straightforward to derive that the index stretch of this attack is $(M - N)m + qm^2 - 2m + 1$. □

Proof of Lemma 4. We prove that adversary M_{out} can get $\mathcal{B}_i$ for $i = 1, 2, \ldots, r$ by mathematical reduction with the following two steps ① and ②. ② Given $i = 1, \ldots, r$, if M_{out} obtains the blocks in $\mathcal{B}_i$, then M_{out} can further get the blocks in $\mathcal{B}_{i-1}$.

① We show how adversary M_{out} acquires the blocks in $\mathcal{B}_r$. Given $x_{A+rqm-r-(\sum_{\ell=1}^{r}\beta_\ell)m+jm}|_{j=0,1,\ldots,q-1}$ and $x_{A+(r+1)qm-(r+1)-(\sum_{\ell=1}^{r}\beta_\ell)m}$, adversary M_{out} can calculate $x_{A+(r+1)qm-r-(\sum_{\ell=1}^{r}\beta_\ell)m}$. Therefore, adversary M_{out} have all the blocks in $\mathcal{B}_r$.

Second, to prove ②, we demonstrate that given $i = 1, \ldots, r$, if M_{out} gets all the blocks in $\mathcal{B}_i$, then M_{out} can also acquire all the blocks in $\mathcal{B}_{i-1}$. We first prove that adversary M_{out} gets $x_{A+iqm-(i-1)-(\sum_{\ell=1}^{i-1}\beta_\ell)m+zm}|_{z=0,1,\ldots,(r+1)q-iq-\sum_{\ell=i}^{r}\beta_\ell}$. also by mathematical reduction with the following two steps ❶ and ❷. ❶ For $z = 0$, adversary M_{out} uses $x_{A+iqm-i-(\sum_{\ell=1}^{i-1}\beta_\ell)m}$ in $\mathcal{B}_i$ and $x_{A+(i-1)qm-(i-1)-(\sum_{\ell=1}^{i-1}\beta_\ell)m+jm}|_{j=0,1,\ldots,q-1}$ to calculate $x_{A+iqm-(i-1)-(\sum_{\ell=1}^{i-1}\beta_\ell)m}$. ❷ Let adversary M_{out} obtain $x_{A+iqm-(i-1)-(\sum_{\ell=1}^{i-1}\beta_\ell)m+zm}|_{z=0,1,\ldots,z_*}$, where $z_* \in \{0, 1, \ldots, (r+1)q - iq - \sum_{\ell=i}^{r}\beta_\ell - 1\}$. From the condition, M_{out} also gets

$$x_{A+(i-1)qm-(i-1)-(\sum_{\ell=1}^{i-1}\beta_\ell)m+jm}|_{j=0,1,\ldots,q-1} = x_{A+iqm-(i-1)-(\sum_{\ell=1}^{i-1}\beta_\ell)m+zm}|_{z=-q,-(q-1),\ldots,-1.}$$

Hence, M_{out} has $x_{A+iqm-(i-1)-(\sum_{\ell=1}^{i-1}\beta_\ell)m+zm}|_{z=-q,\ldots,z_*}$. It's straightforward to see block $x_{A+iqm-i-(\sum_{\ell=1}^{i-1}\beta_\ell)m+(z_*+1)m}$ belongs to $\mathcal{B}_i$. Then M_{out} utilizes $x_{A+iqm-i-(\sum_{\ell=1}^{i-1}\beta_\ell)m+(z_*+1)m}$ and

$$\begin{aligned} &x_{A+iqm-(i-1)-(\sum_{\ell=1}^{i-1}\beta_\ell)m+zm}|_{z=-q+z_*+1,-q+z_*+2,\ldots,z_*}. \\ &= x_{A+(i-1)qm-(i-1)-(\sum_{\ell=1}^{i-1}\beta_\ell)m+(z_*+1)m+jm}|_{j=0,1,\ldots,q-1} \end{aligned}$$

to compute $x_{A+iqm-(i-1)-(\sum_{\ell=1}^{i-1}\beta_\ell)m+(z_*+1)m}$. Owing to ❶ and ❷ above, M_{out} acquires

$$x_{A+iqm-(i-1)-(\sum_{\ell=1}^{i-1}\beta_\ell)m+zm}|_{z=0,1,\ldots,(r+1)q-iq-\sum_{\ell=i}^{r}\beta_\ell.},$$

which with $x_{A+(i-1)qm-(i-1)-(\sum_{\ell=1}^{i-1}\beta_\ell)m+jm}|_{j=0,1,\ldots,q-1}$ also obtained by M_{out} compromise set $\mathcal{B}_{i-1}$.

The result follows given ① and ②. □

Proof of Theorem 2. Let $\mathcal{C}_{ben}$ and $\mathcal{C}_{mal}$ be the computation time for a pool update in the benign case and in the malicious case, respectively.

It is straightforward to derive $\mathcal{C}_{ben} = \lambda(q+1)C_{P_n}$.

Then we compute $\mathcal{C}_{mal}$ in the malicious case.

For $i \geq 0$, we also use C_i to denote the time cost in the malicious case to compute the unmemoized block $g_{nS}(N+c+i)$. For $-N \leq i < 0$, we define C_i as 0 only for ease of notation. Note that for $-N \leq i < 0$, C_i is not the time cost in the malicious case to compute the unmemoized block $g_{nS}(N+c+i)$. We have for $i \geq 0$,

$$C_i = C_{i-1} + \sum_{j=1}^{j=q} C_{i-jm} + (q+1)C_{P_n}. \tag{20}$$

From (20), we obtain $C_i \geq 2C_{i-m}$ for $i \geq m$ and $C_i = (i+1)(q+1)C_{P_n}$ for $0 \leq i \leq m-1$. Then for $i \geq 0$, it follows that $C_i \geq 2^{\lfloor \frac{i}{m} \rfloor}[(i \bmod m)+1](q+1)$.

The total cost $\mathcal{C}_{mal}$ equals the cost to compute $g_{nS}(N+i)|_{0\leq i\leq c-1}$, plus the cost to compute unmemoized blocks $g_{nS}(N+c+i)|_{\max\{\lambda-N-c,0\}\leq i\leq \lambda-1-c}$. Then

$$\begin{aligned} \mathcal{C}_{mal} &= c(q+1)C_{P_n} + \sum_{i=\max\{\lambda-N-c,0\}}^{\lambda-1-c} C_i \\ &\geq [2^{\lceil \frac{\lambda-1-c}{m} \rceil - 2}m(m+1)(q+1) + c(q+1)]C_{P_n}. \end{aligned}$$

□