Low-cost Mitigation against Cold Boot Attacks for an Authentication Token

Ian Goldberg[⋆1], Graeme Jenkinson[2], and Frank Stajano[2]

[1] University of Waterloo (Canada)
[2] University of Cambridge (United Kingdom)

Abstract. Hardware tokens for user authentication need a secure and usable mechanism to lock them when not in use. The Pico academic project proposes an authentication token unlocked by the proximity of simpler wearable devices that provide shares of the token's master key. This method, however, is vulnerable to a cold boot attack: an adversary who captures a running Pico could extract the master key from its RAM and steal all of the user's credentials. We present a cryptographic countermeasure—bivariate secret sharing—that protects all the credentials except the one in use at that time, even if the token is captured while it is on. Remarkably, our key storage costs for the wearables that supply the cryptographic shares are very modest (256 bits) and remain *constant* even if the token holds thousands of credentials. Although bivariate secret sharing has been used before in slightly different ways, our scheme is leaner and more efficient and achieves a new property—cold boot protection. We validated the efficacy of our design by implementing it on a commercial Bluetooth Low Energy development board and measuring its latency and energy consumption. For reasonable choices of latency and security parameters, a standard CR2032 button-cell battery can power our prototype for 5–7 months, and we demonstrate a simple enhancement that could make the same battery last for over 9 months.

Keywords: Hardware authentication token, cold boot attack, memory remanence, bivariate secret sharing, Bluetooth Low Energy

1 Introduction

In 2014 the influential FIDO industry consortium published version 1.0 of its Universal Authentication Framework specification [1], which defines a token-based authentication system intended to replace passwords. The token itself might be unlocked with a variety of methods, such as biometrics. In 2011, Stajano proposed Pico [2], a system with similar goals and architecture (replacing passwords by a token that locally stores a different credential per verifier) but with a proximity-based secret-sharing method for unlocking the token (as originally proposed in 2001 by Desmedt et al. [3]) that would also allow continuous authentication (similar to what Corner and Noble [4] first demonstrated in 2002). In this work we revisit the security of the Pico token-unlocking proposal, improving its resilience against memory readout attacks such as "cold boot" [5].

⋆ On sabbatical at the University of Cambridge while this work was being carried out.

We use the Pico terminology throughout, since this is the reference design that we are improving upon; however, our techniques might also be applied to FIDO UAF or to any other security token that requires locking when not in possession of its user, provided it adopted the proximity-based secret-sharing technique called "Threshold things that think" by Desmedt et al. [3] and "Picosiblings" by Stajano [2]. In this introduction we first set the scene by summarizing the relevant features of Pico. We then explain what additional security benefits we provide with our work.

The Pico is a security token containing hundreds of login credentials, stored in encrypted form in the token's permanent memory. Because the Pico's aim is to allow people to authenticate without having to remember secrets, by design its storage is not unlocked by a PIN or password but rather by the presence of other small wearable devices, the Picosiblings, using a k-out-of-n secret sharing scheme to reconstruct the strong master key that decrypts the Pico's storage.[3] Without the master key, which is not stored in the Pico, the encrypted credentials are unreadable. The intention is that, if adversaries capture the Pico, they will not be able to read its secrets, not even if they also capture a few Picosiblings. No protection is offered by the scheme, though, if the adversary manages to capture the Pico and at least k Picosiblings.

To avoid storing the master key in the Pico, the master key is securely erased immediately after having been reconstructed and used. The decrypted credentials themselves are securely erased after a short interval. Whenever the Pico needs to use any of its credentials (again), the master key must be reconstructed to allow their decryption, but reconstruction can only happen if at least k Picosiblings are within range. Therefore a group of k Picosiblings creates an aura of safety around its wearer, within which the Pico can unlock its credentials.

The privacy of the wearer would be under threat if a passive observer could recognize the Picosiblings by their transmissions, thereby identifying the wearer and tracking their location. The security of the credentials would be under threat if an active attacker could impersonate the target Pico to the Picosiblings and persuade them to release their shares of the master key. For this reason, the communications between a Pico and its Picosiblings must be authenticated and encrypted, and the Picosiblings must only respond to their own Pico.

The primary focus of this paper is to protect the Pico against cold boot attacks. A cold boot attack consists of capturing the running device while plaintext secrets are in RAM, power-cycling it without allowing a clean shutdown and then exploiting data remanence to read its secrets from RAM, as described by Halderman et al. [5]. We generalize the threat model, beyond the literal cold boot, to any other potential means of reading the secrets present in RAM at the time the running Pico is captured, regardless of whether they involve power-cycling the device. The plaintext secrets under threat would include the credentials themselves, the reconstructed master key used to decrypt them and the shares of the master key received from the Picosiblings.

[3] In Pico, for additional security, some special shares are supplied by user biometrics and by a network server [2], and different shares may have different weights [6]. The work described in this paper is independent of these features and therefore for simplicity in what follows we shall ignore these aspects here and treat all shares equally unless otherwise noted.

1.1 Highlights

Although the individual techniques we use (secret sharing, bivariate polynomials, etc.) have been proposed before, our variant is original, the way we combine them is original, and the protection features we achieve have not previously appeared in the literature. We also build a working prototype and measure its performance. In particular:

We design a new internal architecture for the Pico security token and a communication protocol between Pico and Picosiblings that, besides meeting all the security and privacy requirements in the original Pico paper,[4] additionally protect the Pico from cold boot attacks. The hundreds of credentials on the Pico are never all exposed in plaintext, even within the device: the Pico only unlocks the credentials in use at the time. This minimizes exposure to any memory readout attacks, however performed.

We partition the Pico credential database into independently encrypted bins. Crucially, our protection scheme is such that the additional key storage cost per Picosibling is *constant* with respect to the number of bins.

Whether the credentials in the Pico are public keys or symmetric keys [7], we achieve the above without resorting to public-key primitives, so as to facilitate energy-efficient implementation and to avoid introducing a failure point under the threat that quantum computing might one day break today's public-key cryptosystems.

We validate our design with a prototype implementation using commercial off-the-shelf Bluetooth Low Energy parts that demonstrates most of the above features. Working Picosiblings had never been implemented in hardware before and thus our prototype, besides offering enhanced security, is the first concrete instantiation of Picosiblings on which performance measurements can be conducted.

2 The problem

In the original design for Pico, as summarized above, the credential storage is encrypted by a master key that in turn is assembled from cryptographic shares received from the Picosiblings. If adversaries steal a Pico that has been switched off, they will not be able to recover the master key (which only exists in temporary storage), nor any of the user's credentials (stored in permanent storage, but encrypted under the master key), nor any of the received shares (which need to be cached in some readable form until at least k of them have been assembled). The problem, however, is that they might steal a running Pico, with received shares or decrypted secrets in RAM, and somehow find a way of reading these shares or secrets while they lie in RAM in plaintext.

A more subtle problem is that, in order to communicate securely with the Picosiblings, the Pico needs to store, permanently, some Pico-to-Picosiblings communication keys. Since the master key that unlocks the storage of the Pico is reconstructed from shares supplied by the Picosiblings, it is clear that the communication keys to talk to the Picosiblings cannot be encrypted under the master key, or they would be inaccessible when the Pico boots up. But if they are stored in plaintext and accessible to attackers who capture the Pico, then such attackers would be in a position to impersonate the Pico to the Picosiblings and therefore obtain the shares from them. Stannard and

[4] We repeat these requirements in Section 2 for the convenience of the reader.

Stajano [8] acknowledge this threat and invoke some amount of tamper resistance in order to address it. Clearly, if the Pico's processor and memory are all enclosed in a tamper-resistant perimeter, any memory-readout attacks will fall outside the adversary model. However it is hard for us as independent researchers to constructively validate the strength of this defense because tamper-resistant system-on-chip devices are usually only sold to corporations, under NDA and in large volumes.

To protect the security and privacy (including location privacy) of the owner of the Pico, the original design [2] called for a protocol providing the following features:

- The Pico can ascertain the presence of any of its Picosiblings in the vicinity.
- The Picosibling responds to its master Pico but not to any other Pico.
- At each ping, the Picosibling sends its k-out-of-n share to the Pico, in a way that does not reveal it to eavesdroppers.
- An eavesdropper can detect the bidirectional communications between Pico and Picosiblings but not infer identities or long-term pseudonyms.
- The Pico can detect and ignore old replayed messages.
- The Pico can detect and ignore relay attacks (e.g. with Hancke-Kuhn [9]).

Most of these properties were offered in Stannard and Stajano's protocol [8]; however, as our current work redefines the communication between Pico and Picosiblings, these properties must be preserved.

2.1 Attacker model

- We assume the attacker can listen to some of the communications between Pico and Picosiblings, but not to those of their initial setup and pairing ("secure at first use").
- We assume the attacker can send messages to the Pico and/or Picosiblings, but not during initial setup and pairing.
- We assume the attacker can capture and read out the content of a Pico (whether on or off) and fewer than k Picosiblings, but only after initial setup and pairing has taken place.[5]
- As a concession to the considerations above, we assume it is possible for the defender to use low-cost tamper-proofing facilities of the kind in use in smartcards and phone SIMs in order to provide a small amount of permanent storage that the adversary cannot read, even after acquiring physical control of the device.[6] We assume however that the adversary is otherwise able to read out the bulk storage (flash) and workspace (RAM) of the captured device.

The attacker wins if she can extract all the credentials in plaintext out of a captured Pico or if she can use a captured Pico to authenticate as its owner (the former implies the latter).

[5] The number of devices compromised by the attacker can never go down (the attacker cannot "unlearn" the secrets of a device he previously compromised) and any Picosibling that he ever compromised counts towards the quota that cannot reach k in our adversary model.

[6] Note that such protected storage would be needed by Pico to protect the communication keys with the Picosiblings regardless of cold boot protection, as acknowledged by Stannard and Stajano [8].

3 Our solution in a nutshell

Our core idea is to partition the encrypted storage of the Pico into many small bins, each holding just a few credentials (ideally just one, subject to technical limitations), and to redesign the secret sharing scheme and the communication protocol with the Picosiblings so that only one bin gets decrypted at a time.

Instead of one master key for the whole Pico we now have as many master keys as there are bins. These are, again, reconstructed from shares supplied by the Picosiblings, but now the Pico must first ask for the shares that are relevant to a particular bin.

We keep the communication keys with the Picosiblings in tamper-resistant storage from which the adversary cannot economically extract them. (We could conceivably also store a "master key to encrypt the whole RAM" in there as well; however, in the absence of a dedicated cryptoprocessor, the decrypted RAM, or at least the current block of decrypted RAM, would then have to be written out in some workspace in order to be used, and we have assumed that the adversary could access that.)

This means that even an adversary who can read the memory of our Pico (except for the Picosibling communication keys held in tamper-resistant storage) can at most acquire the credentials of one of the bins. We therefore protect the security token from cold boot attacks even when it is not possible (for reasons of cost, performance or simply because the manufacturers will not sell us any potentially suitable hardware) to resort to enclosing the whole processor and RAM into a tamper-resistant enclosure.

The trade-off of our approach is that, whereas previously when the master key had been reconstructed all the credentials were available instantly, here the Pico must first decide which bin of credentials to decrypt, then request the shares of the relevant bin key from the nearby Picosiblings and only then, after a round trip, will it be able to decrypt the relevant credentials. This introduces latency. User acceptability of an alternative security mechanism is largely unaffected by the level of security it offers (which is unobservable by most users anyway) but is dramatically affected by waiting time. For this reason, besides developing the crypto, we implemented our system on commercial off-the-shelf Bluetooth LE development boards in order to measure whether the latency and power consumption of our approach would be acceptable in a realistic setting.

4 A new secret sharing scheme for authentication tokens

4.1 The Pico credential database

A Pico stores a potentially large number N of user credentials, grouped into 256 bins.[7] The value 256 is a parameter of the system; it is straightforward to make it larger, at the cost of somewhat increased implementation complexity. For each account the user has, there is one *(userid, credential)* pair. The user may, but does not have to, use a different userid for every account. The user may also have several accounts with the same service, obviously with distinct userids.

[7] If $N \leq 256$ then each credential has its own bin and can be decrypted independently of the others. If $N > 256$ then some bins may contain more than one credential, which will be encrypted and decrypted together.

The original Pico design [2] was vulnerable to a cold boot attack: the entire credential database was encrypted with a single master key, shared across the Picosiblings using Shamir secret sharing [10]. While the Pico was being unlocked, that key, and so the entire set of credentials, could be exposed. To mitigate this attack, in our design we encrypt the credentials in each bin with their own *bin key*.

Doing this in a naive way, however, would require the Picosiblings to each store shares of hundreds of keys, which is far too much—we want the Picosibling storage requirements to be very small. To this end, we use a *keying polynomial* $K(y)$ of degree r, and set the encryption key for bin β to $K(\beta)$. The keying polynomial $K(y)$ is shared to the Picosiblings in the manner described in Section 4.2 below.

The choice of the polynomial degree r is important: the Picosiblings will each have to store $r+1$ key-sized entries, so we would like to keep r small. However, if the Pico is ever actively unlocking *more than r bins at the same time*, then at least $r+1$ values of $K(\beta_i)$ will be in memory simultaneously. A cold boot attack at that point will be able to recover the entire polynomial K, and thus decrypt *all* of the bins. Note that this is not to say that credentials from at most r bins can be unlocked at any time in the Pico; once the Pico unlocks a bin (reconstructs the bin key $K(\beta_i)$ and decrypts the desired credential(s) in the bin), the Pico will wipe the key (and the shares that constructed it) from memory. It is only if the Pico is *actively receiving shares* of more than r bin keys at the same time that the fatal problem arises.

We can therefore choose r to be quite small, and simply program the Pico to never request more than r key reconstructions in parallel. Indeed, $r = 1$ is a perfectly reasonable choice, and results in each Picosibling having to store only two key-sized values—256 bits in total. (Note that $r = 0$ corresponds to the original Pico strategy [2] of encrypting every credential with the same key.)

Appendix A presents the schema for the Pico credential database.

4.2 Bivariate secret sharing

The secret to be shared across the Picosiblings (say there are n Picosiblings) is the above keying polynomial, which is an arbitrary degree-r polynomial $K(y) = \sum_{j=0}^{r} k_j y^j$.

The possible inputs y for this polynomial are bin identifiers (β_i), which are values in a small finite field $\mathbb{F}$. In our implementation, we choose $\mathbb{F} = GF(2^8)$, so the number of bins is $|\mathbb{F}| = 256$, and each bin identifier is a single byte. However, the outputs of the polynomial should be encryption keys, which should of course be much larger than 8 bits. Therefore, we select the coefficients k_j of the keying polynomial from a *vector space* $\mathbb{V}$ over $\mathbb{F}$; in particular, we choose $\mathbb{V} = \mathbb{F}^{16}$, so that the elements of $\mathbb{V}$ are vectors (arrays) of 16 bytes (128 bits). Thus we write $K(y) \in \mathbb{V}[y]$.

In order to share an entire polynomial $K(y)$, rather than a single encryption key as in the original design, we now have the Pico create a *bivariate polynomial F(x,y)* of degree $(k-1, r)$—that is, of degree $k-1$ in x and of degree r in y: $F(x,y) = \sum_{i=0}^{k-1} \sum_{j=0}^{r} a_{ij} x^i y^j$. For a univariate polynomial, k points $y_i = f(\alpha_i)$ define a unique polynomial f of degree $k-1$. The equivalent statement for a bivariate polynomial is that k univariate

polynomials $f_i(y) = F(\alpha_i, y)$ of degree r define a unique bivariate polynomial F of degree $(k-1, r)$.

A bivariate secret sharing scheme for n participants (the Picosiblings, in our case) is defined as follows. Let $\mathbb{F}$ be a finite field; $\mathbb{V}$ be a vector space over $\mathbb{F}$; k, r, and n be non-negative integers with $1 \leq k \leq n$; and $\alpha_1, \ldots, \alpha_n$ be arbitrary distinct non-zero elements of $\mathbb{F}$. (The α_i are the Picosibling identifiers selected by the Pico. Although our protocol never requires these identifiers to leave the Pico—even the Picosiblings never learn their own identifiers—they are not security sensitive, in the sense that their knowledge would not help an adversary guess any of the shares.) As above, the secret to be shared is the univariate polynomial $K(y) = \sum_{j=0}^{r} k_j y^j \in \mathbb{V}[y]$.

For $0 \leq j \leq r$, set $a_{0j} = k_j$, and for $1 \leq i \leq k-1$ and $0 \leq j \leq r$, select a_{ij} uniformly at random from $\mathbb{V}$. Then construct the bivariate polynomial $F(x, y) \in \mathbb{V}[x, y]$ as above.

For each $1 \leq i \leq n$, compute the degree-r polynomial $f_i(y) = F(\alpha_i, y) \in \mathbb{V}[y]$, and send $f_i(y)$ (the *share*) to participant i. Note that the amount of storage this requires at each participant is $r + 1$ elements of $\mathbb{V}$.

In a typical secret-sharing protocol, k participants would combine their shares to recover the shared secret *polynomial* $K(y)$. Our scenario is slightly different, however; for a specified bin identifier β, we wish to reconstruct just the single *value* $K(\beta) \in \mathbb{V}$, and not the whole polynomial $K(y)$. To accomplish this reconstruction, we will send the value β to k Picosiblings. Each Picosibling i will reply with $v_{\beta i} = f_i(\beta) = F(\alpha_i, \beta)$—a single value in $\mathbb{V}$. We then perform Lagrange interpolation on the $(\alpha_i, v_{\beta i})$ pairs in the usual way to recover $F(0, \beta) = K(\beta)$.

We can achieve proactivity [11] by periodically creating n shares of the zero polynomial, and sending them to the Picosiblings to add to their existing shares (queueing them on the Pico until the next time each Picosibling is encountered—this is safe, since the queued value reveals no information about either the old or new value of the share of the keying polynomial). To remove a Picosibling from the scheme, the same mechanism is used, except the removed Picosibling is not sent a share of the zero polynomial. To add a Picosibling to the scheme, k existing Picosiblings get together to reconstruct F, a new and unused $\alpha_{n+1} \neq 0$ is picked, and $F(\alpha_{n+1}, y)$ is sent to the new Picosibling.

4.3 Picosibling protocol

We now develop a protocol based on the above bivariate secret sharing scheme that allows individual bins in the Pico credential database to be unlocked as needed.

Enrollment When a new Picosibling is enrolled to a Pico, the Pico executes a pairing protocol with the Picosibling in order to establish a random shared symmetric communication key CK_i that will be used to protect all communication between the Pico and that Picosibling. The pairing mechanism is outside the scope of the current discussion; see Krause [12] for a description of the proposed pairing mechanisms for Picos and Picosiblings. At pairing time, the Pico also selects an arbitrary unused non-zero $\alpha_i \in \mathbb{F}$ to serve as that Picosibling's *Picosibling identifier*. The Pico will store each of the communication keys CK_i in its small tamper-proof memory, while the α_i are

not sensitive, and need not be protected in this manner. If the tamper-proof memory is extremely tight, the CK_i can each be derived from a single master 128-bit secret CK^* as $CK_i = \text{CBC-MAC}_{CK^*}(\alpha_i)$; in that case, only the 128-bit CK^* needs to be stored in tamper-proof memory.

The first batch of at least k Picosiblings will be enrolled at the time the Pico is initialized. At this time, the Pico will create the keying polynomial $K(y)$, use it to encrypt the credential database, and send shares of $K(y)$ to the Picosiblings as described above. Later, new Picosiblings can be enrolled by having the Pico communicate with k existing Picosiblings to reconstruct the entirety of the bivariate polynomial $F(x, y)$ that produces the shares of the keying polynomial $K(y) = F(0, y)$. The Pico will then send the new Picosibling the coefficients $f_{i0}, f_{i1} \in \mathbb{V}$ of its share of the keying polynomial.

Query share When the user attempts to authenticate to a service, the Pico will look up the bin identifier β associated with the credential for that service in its credential database. The Pico will then attempt to construct the bin key $K(\beta)$ by communicating with each of k nearby Picosiblings. This communication will be protected using the above symmetric communication keys CK_i.

The Pico sends the value β to each Picosibling; the Picosibling evaluates its share $v_{\beta i} = f_{i0} + \beta * f_{i1}$ of the keying polynomial for the specified bin and returns the value $v_{\beta i}$ to the Pico.

The Pico (which knows the value of α_i associated with each Picosibling) then uses Lagrange interpolation to find the bin key $K(\beta) = \sum_{i=1}^{k} v_{\beta i} \prod_{\substack{1 \le j \le k \\ j \ne i}} \frac{\alpha_j}{\alpha_j - \alpha_i}$. The bin key is then used to decrypt the desired credential in bin β of the Pico credential database. Once the credential is accessed, the shares and the reconstructed $K(\beta)$ can be deleted from memory.

The bin key reconstruction process is when the Pico is at its most vulnerable: if it has received some, but not all, of the k shares it has requested, then an adversary cold booting the Pico at that point could recover some shares of the bin key, and capturing up to $k - 1$ Picosiblings (which is within the threat model) would reveal the bin key to the adversary. However, unlike the original Pico design, *only* the credentials in that bin are revealed, and not all credentials in the Pico. One might be tempted to use a technique such as that of TRESOR [13] to do the entirety of the bin key reconstruction and bin decryption in CPU registers; being able to do so would add even more security to our proposal. We could not experiment with this idea because in our prototype the role of the Pico was played by a BLE-capable Android phone whose non-Intel CPU did not support the AES-NI instruction set required by TRESOR.

Query presence Once the credential has been used, the Pico enters into a continuous authentication mode with the service. [2] The continuous authentication uses an ephemeral key and, therefore, no longer needs access to the credential database. However, if the Pico is out of range of its siblings it automatically locks and pauses any active sessions.

While the Pico is in continuous authentication mode, it periodically sends heartbeat requests to its nearby Picosiblings. As long as k Picosiblings respond to the heartbeat, the Pico will maintain continuous authentication to the service. These siblings do not necessarily have to be the devices that initially provided shares to unlock the credential, although it is reasonable for the Pico to try to contact those devices first.

5 Prototype implementation

To validate and measure the performance of our cryptographic design, we implemented our scheme on a realistic hardware development platform for wearable devices.

To our knowledge, this is the first time that working Picosiblings as dedicated devices have been prototyped. Our self-imposed implementation constraints attempt to limit the burden imposed on the user in various dimensions (size, weight, maintenance, obtrusiveness, cost, etc.). More specifically, Picosiblings should:

- Be small enough to be attached (unobtrusively) to a range of items that users already frequently carry (such as wallets, phones, and keys).
- Be able to be integrated into items that users carry or wear.
- Operate for many months without charging or replacing batteries.
- Be cheap to purchase and replace.

Whilst somewhat vague, these non-functional requirements introduce a set of constraints against which the success of our prototype implementation can be measured.

The Pico and its Picosiblings are heavily asymmetric, with Picosiblings being both heavily constrained (in terms of size and therefore battery capacity) and poorly resourced (in terms of memory, computation power, and user interface features). Recognizing this, our implementation is optimized around the most resource-constrained devices—the Picosiblings.

A defining architectural choice for Picosiblings is the wireless communication protocol. Low-cost, low-power wireless communication is a key enabling technology for the Internet of Things (IoT). Of the numerous wireless standards employed in IoT applications, Bluetooth Low Energy (BLE) is the closest match to Pico's requirements. BLE is specifically targeted at scenarios exhibiting significant asymmetry, and has been designed around the performance of off-the-shelf button-cell batteries. BLE also supports a privacy feature, in which hardware addresses are encrypted over the air, in order to avoid tracking of devices [14]; this feature also meshes well with our requirements.

5.1 BLE Picosibling service

In contrast to Bluetooth classic (essentially a cable replacement), BLE is targeted at applications that transmit only a few octets of data at frequencies ranging from once a second to every few days, weeks, or months. Such applications typically send only a limited range of primitive data types; for example, representing the value of a temperature sensor. In BLE, server state is made available through sets of *characteristics*. Characteristics group together state (both readable and writable) and metadata such as

name and access permissions. Readable characteristics expose the historic or current state recorded by the service. For example, a simple light switch service may expose the state of the light (either on or off). Writable characteristics are commonly used to command behaviour; for example, turning a light on or off. Characteristics and associated behaviours are grouped together into reusable components called services. Clients can interrogate a service's characteristics and associated metadata to determine how to interact with the service; this can be viewed as a implementation of the well-known Service Oriented Architecture (SOA) pattern:

> A paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. It provides a uniform means to offer, discover, interact with and use capabilities to produce desired effects consistent with measurable preconditions and expectations. [15]

The following sections provide an overview of the BLE Picosibling service.[8]

Enrollment To enroll a new Picosibling, it must be first awoken and made discoverable by the Pico. In our prototype implementation, unenrolled Picosiblings are woken up by the user pressing a button. However, as the design matures, such UI elements will be removed. As with commercial BLE beacon devices, awakening the device from sleep will be performed by the user tapping the device (which is detected by a MEMS accelerometer). Once awake, the Picosibling enters the BLE general advertising mode. Whilst in this mode, the Picosibling can accept a connection event from the Pico. Once connected, the Picosibling's exposed state can be read and written.

With weaknesses in BLE pairing mechanisms well documented [16], enrollment of Picosiblings is based upon the principle of "secure at first use". To enroll a new Picosibling, the Pico writes into it its unique communication key CK_i and share $f_i(y)$. The Picosibling's communication key is exposed by the BLE service as a single 16-byte (128-bit) write-only characteristic. When this characteristic is written, the Picosibling performs key diversification[9] to generate a further two keys. These keys provide authenticated encryption (encrypt-then-MAC) for messages sent over the air. Transmitted messages contain counters to prevent reflection and replay attacks.

As noted in Section 4.2, storage for the share requires $r + 1$ elements of $\mathbb{V}$. In this case, values in $\mathbb{V}$ are vectors (arrays) of 16 elements of $\mathbb{F}$; that is, arbitrary 16-byte (128-bit) values. Thus, the share $f_i(y)$ is represented in the BLE profile as a single 32-byte write-only characteristic.

Query share/presence To query the Picosibling's share, the Pico writes the value β (encrypted and MAC'd using the keys established at enrollment) into a write-only characteristic. This characteristic is 48 bytes in length, with 16 bytes for each of the initialization vector (IV), encrypted payload, and a message authentication code (MAC). On

[8] This description is intended to capture the essential features of our implementation rather than act as a formal specification.

[9] The communication key is diversified by performing a CBC-MAC (using the AES coprocessor) on two fixed values (1 and 2). The Picosibling does not possess a source of cryptographically strong randomness, and therefore is not trusted to generate random keys.

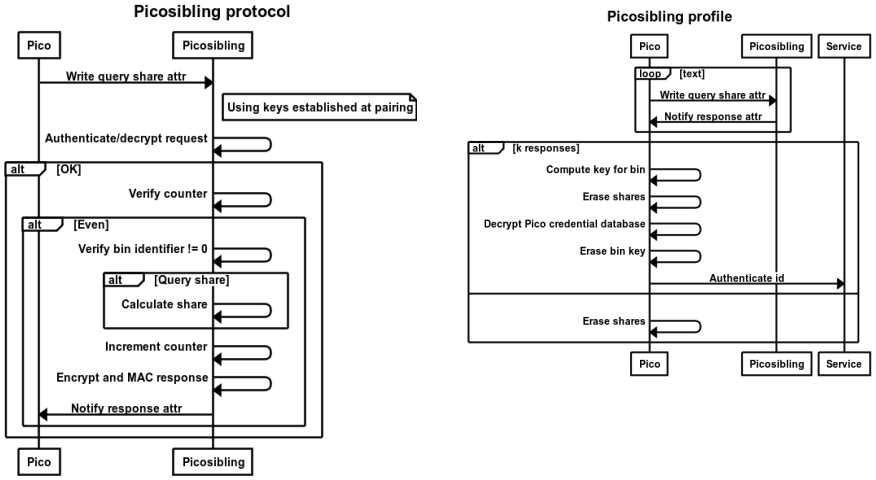

Fig. 1. Overview of the Picosibling protocol implementation (left) and the Picosibling profile implementation (right) in BLE, as UML sequence diagrams.

writing the characteristic, the value is decrypted and verified using the keys established at enrollment. The decrypted payload is a binary packed data structure containing a counter (whose value is even for communication from the Pico to the Picosibling, and odd in the opposite direction), a share flag (set to 1 when the Pico is querying the value of a share, and 0 when it is merely querying presence), and an 8-bit bin identifier β.

Using the bin identifier β, the Picosibling evaluates its share $f_i(\beta)$.[10] The Picosibling share (encrypted and MAC'd) is returned to the Pico by updating a 64-byte read-only characteristic (16 bytes for the IV, 32 bytes for the encrypted payload, including the share and the incremented counter, and 16 bytes for the MAC). Changes to this value are automatically reported as a BLE notification to the Pico.

Querying the Picosibling's presence is achieved as described above, querying the Picosibling while setting the share flag to 0.

Figure 1 (left) shows an overview of the BLE Picosibling profile query share and presence behaviour.

Our prototype Picosiblings have been developed using an off-the-shelf BLE development kit from Texas Instruments (TI), described in Appendix B.

5.2 BLE Picosibling profile

A BLE profile specifies the behaviour of the client (in our case, the Pico). Figure 1 (right) gives an overview of the client behaviour when querying the Picosiblings' shares.

[10] As an optimization, computation in $GF(2^8)$ is performed with two precomputed 256-byte tables. The first provides provides a lookup $i \mapsto g^i$ and the second $g^i \mapsto i$ for a generator g. Note that although the CC2541 device contains 256 KB of flash, there is not a free 64 KB segment capable of holding a 256 * 256 B lookup table required for precomputing the entire multiplication table in $GF(2^8)$.

Querying presence is largely the same, except that if k responses are not received, the Pico deletes its continuous authentication keys and locks up, preventing the user (or a malicious actor) from using the device to authenticate.

Neither BLE nor any of the other commercial wireless communications standards attempts to mitigate against relay attacks. As relay attacks against wireless communications are not (currently) viewed as being widely exploitable, commercial manufacturers find little justification to complicate their implementations by inclusion of complex distance bounding protocols. We therefore lower our expectations slightly:

– Pico can detect and ignore relay attacks *that route messages over the Internet* (where the introduced latencies are sufficiently large to be detected).

To mitigate this attack the Pico client is responsible for measuring round trip times, ensuring that communication with the Picosiblings has not been routed through the Internet.

In common with previous work [17], our Pico client is developed as an Android mobile phone application, which provides the *Quasi-Nothing-To-Carry* property of the evaluation framework defined by Bonneau et al. [18]. Our minimal implementation of Pico supports:

– Requesting Picosibling shares to unlock entries in the credential database (see Appendix A) and
– querying Picosiblings for user presence.

No other features of Pico are implemented in our prototype client.

6 Performance evaluation

In contrast to typical BLE applications such as remote sensing, where communication is infrequent and therefore battery life is measured in months or even years, Pico continually communicates with its Picosiblings to verify that the user, once authenticated, remains present. Thus a key objective of prototyping is to estimate the impact that the protocol of Section 4.3 has on the Picosibling's battery life. Following the detailed guidance given in TI Application Note AN092 [19], we have produced estimates of battery lifetime under representative conditions with the focus on assessing the broad feasibility of our architectural choices.

As a first step, we derive an upper bound estimate on battery lifetime. This corresponds to an active but unused Picosibling; that is, a Picosibling that the user is carrying with them but is not one of the k-out-of-n used by Pico when querying a share or user presence. In contrast to Bluetooth classic, BLE is a connectionless protocol. As a connection-oriented channel is never established, there is little cost in dropping and re-establishing connections only when there is useful data to send [20]. When not communicating, Picosiblings enter a low-power sleep mode.[11] Although Picosiblings spend

[11] As detailed in Kamath and Lindh [19], when sleeping the device enters Power Mode 2 where the current consumed is 1 µA.

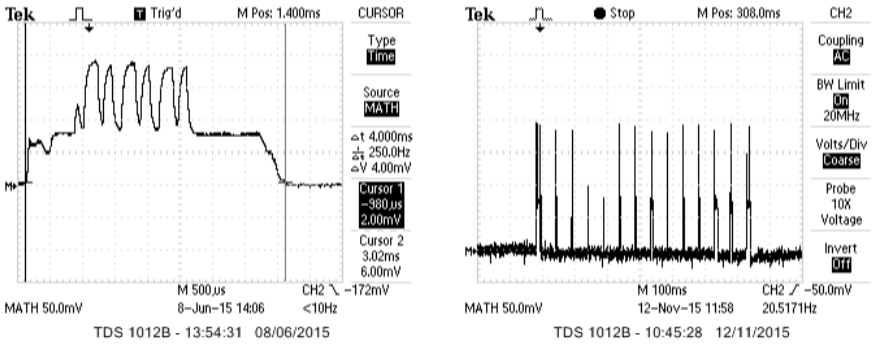

Fig. 2. Left: Oscilloscope trace of current consumed by BLE Picosibling during periodic wakeup, advertising its presence. Right: Oscilloscope trace of the current consumed by a BLE Picosibling when the Pico is querying its presence. (Note the differences in the horizontal scale.)

the majority of their time asleep, they periodically wake up, advertising their presence to respond to requests from the Pico device. The time between wakeups, t_w, is a parameter of our system. As mentioned in Section 3, this time controls how long (after the user initiates a login to a service) the Pico may have to wait before being able to contact its Picosiblings, and thus complete the login. We want this latency to be low, but frequent wakeups will decrease battery life. We analyze this tradeoff next.

Figure 2 shows an oscilloscope trace of current consumption[12] during the period in which the Picosibling is advertising its presence to the Pico device. As is evident from Figure 2, current consumption varies markedly as the Picosibling transitions between different operating states (notably transmitting and receiving, but also waking up and preparing for sleep).

We measured the average current during the advertising period to be 10.42 mA. The time between advertising events is t_w; the advertising period is 4.02 ms, with the Picosibling being asleep (and drawing 1 µA) the remainder of the time. Thus, we estimate the total average current of an active but unused Picosibling as $\frac{((t_w - 4.02\,\text{ms}) * 1\,\mu\text{A} + (4.02\,\text{ms} * 10.42\,\text{mA}))}{t_w} = 1\,\mu\text{A} + \frac{41.9}{t_w}\,\mu\text{A s}$.

In the original design [2], Picosibling shares are used to construct a single full-disk encryption (FDE) key. This FDE key protects the Pico credential database when the user is not present. In contrast, our scheme requires that decryption keys are reconstructed on demand; that is, when the user authenticates with a given service. As keys are reconstructed in the flow of the overall authentication process, response latency is critical. Sasse et al. [21] report that average completion time for username and password authentication is on the order of ten seconds.[13] In our prototype implementation,

[12] Measured as the voltage across a 1 KΩ resistor.

[13] This value was produced by applying the KLM-GOMS methodology (a modelling approach for predicting how long it takes an expert user to accomplish a task on a computing system) [22].

we chose $t_w = 1$ s, so that the Pico reconstructs the encryption keys within one second under ideal conditions and within two to three seconds with some radio interference. This process involves little cognitive effort from the user and is therefore unlikely to be seen as a major burden. This value of t_w yields an average current of about $43\,\mu$A.

Assuming 55 mA h is the capacity of a CR1616 battery (see Appendix B for hardware details), that gives an expected battery lifetime of 55 mA h / 43 µA = 1300 hours, or 53 days. Assuming the capacity of a CR2032 battery is 230 mA h, this results in an expected battery lifetime of 230 mA h / 43 µA = 5300 hours, or 220 days.

Our estimates indicate that an active but unused Picosiblings can operate on a CR2032 battery for over seven months. A Picosibling interacting with a Pico will, however, of course have a somewhat lower battery life; we analyze this effect next. Figure 2 shows the current consumed by the Picosibling when the Pico is querying its presence. Querying presence requires the client (Pico) to write a single BLE characteristic, with the response returned via a BLE notification. However, the captured trace exhibits multiple interactions between the Pico and the Picosibling. Whilst additional messages (for example for setting up the BLE connection) are expected, the sheer number of interactions was somewhat confusing. On detailed investigation we determined that the TI BLE stack limits the MTU of user data to 20 bytes (the minimum allowable in the BLE specification). Thus, when writing or reading characteristics greater than 20 bytes, the communication is broken into multple packets. As the TI BLE stack is provided as a binary library, this parameter could not be changed and the presence of these additional messages is a limitation of our current implementation. We could improve on this, and thus save time and energy, but only if we had the ability to modify the library.

Figure 2 shows that the Picosibling responds to a query in about 680 ms. This could be significantly reduced, saving energy, if more data could be sent in each packet. Note that querying a Picosibling's share requires exactly the same set of interactions with the Pico, and therefore takes approximately the same time. The additional processing time required to evaluate the share is tiny in comparison.

Once a Pico is unlocked, it enters continuous authentication mode. In this mode, the Pico polls its Picosiblings to ensure that at least k of them remain in the Pico's vicinity. How often this polling is done is governed by t_c, another parameter of our system. Again, a higher t_c will increase the battery life. This time, the tradeoff is not to interactive latency, however, but to how long a Pico will stay unlocked after leaving the aura of its Picosiblings. (Note that t_c must be a multiple of t_w, as communication can only occur when the Picosibling wakes up.)

Similar to the above, we compute the average current for a Picosibling undergoing continuous authentication to be $\frac{823}{t_c}$ µA s. This is in addition to the above cost of waking up every t_w, so the total average current is $1\,\mu\text{A} + \frac{41.9}{t_w}\,\mu\text{A s} + \frac{823}{t_c}\,\mu\text{A s}$. (Although, again, the numerator 823 could be significantly smaller with an improved BLE stack with a higher MTU.)

If we retain $t_w = 1$ s, and set $t_c = 30$ s, then the average current is 70 µA. If we are satisfied with $t_c = 60$ s, the average current is 57 µA. With a CR1616 battery, the latter figure would give an expected battery lifetime of 960 hours, or 40 days; with a CR2032 battery, the battery life would increase to 4000 hours, or over 165 days. This value is

within the performance requirements for Picosiblings outlined in Section 5 (and would be even higher with a better BLE stack).

It should also be noted that, under normal circumstances, Pico automatically logs users out of services when they are away from the terminal used to access those services. Thus, the Pico does not need to confirm the presence of its Picosiblings continuously. If a user, for example, is actively logged into a service only half of the time, the lifetime of a CR2032 battery would be over 190 days with our current implementation.

Additionally, the Picosiblings could use a MEMS accelerometer to notice that they are not being worn (when, for example, the user is asleep), and go to sleep themselves for that entire time, without waking up every t_w.

Extending the above example, if the user sleeps 8 hours per day, and is actively logged into a service for half of her waking hours, the lifetime of a CR2032 battery would be over 280 days.

7 Related work

The most obviously related work is Pico: the relevant papers from Stajano and his group [2, 6–8, 17] have been extensively referred to throughout the text.

Laurie and Singer [23] argue that it is impossible to have a system which is both general purpose and trustworthy. Their requirements for a trusted authentication device (which they refer to as the Neb) closely match those of Pico. However, in contrast to Pico, the Neb does not attempt to mitigate loss or theft of the device.

FIDO (Fast IDentity Online), an open industry alliance of vendors, including a who's who of major players such as Alibaba, American Express, ARM, Google, Intel, Mastercard, Microsoft, PayPal, Samsung and Visa, released two sets of specifications for online authentication: UAF (Universal Authentication Framework) [1] and U2F (Universal Second Factor) [24]. Both involve authenticating to online services with a device. UAF replaces passwords entirely by allowing users to authenticate from a FIDO-enabled device, such as a smartphone. It involves registering the user's device to online services and selecting a biometric authentication action, such as swiping a finger, performed on the device. Pico and FIDO share commonalities in both the problem they are addressing and their approaches, though it should be noted that Pico predates the FIDO specifications by over 3 years.

Müller et al.'s TRESOR [13] is closely related to our present work in purpose: it too is a low-cost protection against cold boot attacks. Their approach is totally different, however: it works by running the encryption fully within the CPU registers, with creative use of the debug registers as crypto storage. We considered how to overcome the restrictions on the limited register space available, and we would have liked to offer defense in depth by combining their approach with ours, but were not able to do so because TRESOR is specific to Intel processors supporting the AES-NI instruction set.

There is prior art on sensing the proximity of a user-worn tag to infer user presence and lock and unlock devices, from Want et al.'s Active Badge [25], through Landwehr's patent [26, 27], and to Corner and Noble's "Zero-interaction authentication" [4], the first system in which the user-worn token provided the cryptographic key to unlock the target device. Sharing the key among multiple devices was first suggested by Desmedt

et al. [3]. Peeters defended a PhD [28] on this topic and, among other contributions, showed how to securely store a share in a wearable device that does not offer a secure storage hardware primitive [29]. We did not use it because capturing enough wearables was outside our threat model, but this is another result that might be profitably combined with our design to provide defense in depth.

Bivariate secret sharing has been used in prior works; for example, by Cachin et al. [30] and by Tassa and Dyn [31]. In those works, the secret to be shared was a scalar, and the bivariate polynomial was used to effectively make *shares of shares* of the secret. In contrast, in our work, the secret to be shared is itself the univariate keying polynomial $K(y)$. This polynomial is not acting in a secret-sharing capacity; $K(0)$ is not special—it is merely the bin key for bin number 0.

Proactive secret sharing was introduced by Herzberg et al. [11], and mobile proactive secret sharing (MPSS) by Schultz et al. [32]. Those schemes require relatively heavyweight commitment and public-key primitives. Our system does not require such primitives, because our threat model is less general. In particular, in our setting, the *dealer* (the entity creating the shares from the secret) is the Pico itself, which is assumed to be trusted—there is no need for the Picosiblings to verify that the shares they receive are consistent. Similarly, although we allow for a mobile adversary that can compromise Picosiblings over time, we assume that once an adversary compromises a Picosibling, it cannot be "uncompromised"; that is, the set of compromised Picosiblings is non-decreasing. This allows us to use relatively simple protocols for Picosibling addition and removal, as we do not need to deal with cases where the adversary compromises $k-1$ Picosiblings before a removal, and a *different* $k-1$ Picosiblings afterwards. (MPSS, on the other hand, does go to great effort to deal with such cases.)

8 Conclusions

The remarkable industry momentum gathering behind the FIDO alliance suggests a strong convergence towards replacing passwords with a personal authentication token. If that premise is accepted, the next problem is how to lock and unlock the token to protect it against unauthorized use. While FIDO UAF [1] offers static unlocking, Pico [2] adopts the proximity-based threshold scheme pioneered by Desmedt et al. [3], using it as a platform for continuous authentication.

We found Pico vulnerable to a cold boot attack: an adversary who captured a running Pico and was able to read its RAM would be able to steal all the credentials of the user. We designed and prototyped a low-cost mitigation for this vulnerability, without resorting to the silver bullet of making the whole security token tamper-proof. Bootstrapping all security from a small amount of tamper-resistant flash memory (which Pico requires anyway, regardless of our contribution), we protect the user's credentials against an adversary who may read out all the flash and RAM of a running token. Our new secret sharing scheme, based on a bivariate polynomial, allows only a small fraction of the credentials to be exposed at a time. Our solution scales to a high number of credentials while imposing only a constant (and small) storage cost on the Picosiblings.

We have prototyped our cryptographic design on COTS hardware and measured its performance in terms of power consumption and latency. Our Picosibling prototype

lasts for 165–220 days (depending on how frequently the user is authenticated to a service) on a CR2032 button-cell battery, with an observed authentication latency of 2–3 seconds. We also show how a simple accelerometer detecting when the Picosibling is not being worn could increase the battery life by about 50%. Fixing gratuitous inefficiencies in the BLE stack would increase it further.

We trust readers will agree that our prototype, though not yet ready for prime time, soundly demonstrates the viability of our design: if it were made into a commercial product, battery life and latency could be improved even further.

Acknowledgements We thank Rob Harle for his advice on selecting a Bluetooth Low Energy development platform. We thank David Llewellyn-Jones for insightful comments on the comparative security of our scheme with respect to related work. Jenkinson and Stajano thank the European Research Council for funding this research through grant StG 307224 (Pico). Goldberg thanks NSERC for grant RGPIN-341529.

References

1. FIDO Alliance: FIDO UAF complete specifications FINAL 1.0 (December 2014)
2. Stajano, F.: Pico: no more passwords! In: Proceedings of the 19th international conference on Security Protocols. SP'11, Berlin, Heidelberg, Springer-Verlag (2011) 49–81
3. Desmedt, Y., Burmester, M., Safavi-Naini, R., Wang, H.: Threshold Things That Think (T4): Security Requirements to Cope with Theft of Handheld/Handless Internet Devices. In: Proc. Symposium on Requirements Engineering for Information Security. (2001)
4. Corner, M.D., Noble, B.D.: Zero-interaction authentication. In: Proc. ACM MobiCom 2002, New York, NY, USA, ACM (September 23–28 2002) 1–11
5. Halderman, J.A., Schoen, S.D., Heninger, N., Clarkson, W., Paul, W., Calandrino, J.A., Feldman, A.J., Appelbaum, J., Felten, E.W.: Lest we remember: Cold-boot attacks on encryption keys. Commun. ACM **52**(5) (May 2009) 91–98
6. Stafford-Fraser, Q., Stajano, F., Warrington, C., Jenkinson, G., Spencer, M., Payne, J.: To have and have not: Variations on secret sharing to model user presence. In: Proc. UPSIDE workshop of UBICOMP 2014. (September 2014)
7. Stajano, F., Christianson, B., Lomas, M., Jenkinson, G., Payne, J., Spencer, M., Stafford-Fraser, Q.: Pico without public keys. In et al., B.C., ed.: Proc. Security Protocols Workshop 2015. Volume 9379 of LNCS., Springer (April 2015) 172–186
8. Stannard, O., Stajano, F.: Am I in good company? A privacy-protecting protocol for cooperating ubiquitous computing devices. In Bruce Christianson et al., ed.: Proc. Security Protocols Workshop 2012. Volume 7622 of LNCS., Springer (April 2012) 223–230
9. Hancke, G.P., Kuhn, M.G.: An RFID Distance Bounding Protocol. In: Proc. IEEE SECURECOMM 2005, Washington, DC, USA, IEEE Computer Society (2005) 67–73
10. Shamir, A.: How to Share a Secret. Commun. ACM **22**(11) (1979) 612–613
11. Herzberg, A., Jarecki, S., Krawczyk, H., Yung, M.: Proactive Secret Sharing Or: How to Cope With Perpetual Leakage. In: Advances In Cryptology—CRYPTO '95, Springer (1995) 339–352
12. Krause, F.M.A.: Designing Secure & Usable Picosiblings: An exploration of potential pairing mechanisms. Master's thesis, Wolfson College, University of Cambridge (2014)
13. Müller, T., Freiling, F.C., Dewald, A.: TRESOR Runs Encryption Securely Outside RAM. In: 20th USENIX Security Symposium, USENIX (2011)

14. Gomez, C., Oller, J., Paradells, J.: Overview and Evaluation of Bluetooth Low Energy: An Emerging Low-Power Wireless Technology. Sensors **12**(9) (2012) 11734–11753
15. MacKenzie, C.M., Laskey, K., McCabe, F., Brown, P.F., Metz, R., Hamilton, B.A.: Reference model for service oriented architecture 1.0. OASIS Standard **12** (2006)
16. Ryan, M.: Bluetooth: With Low Energy Comes Low Security. In: 7th USENIX Workshop on Offensive Technologies, Berkeley, CA, USENIX (2013)
17. Stajano, F., Jenkinson, G., Payne, J., Spencer, M., Stafford-Fraser, Q., Warrington, C.: Bootstrapping adoption of the pico password replacement system. In: Security Protocols XXII. Springer (2014) 172–186
18. Bonneau, J., Herley, C., van Oorschot, P.C., Stajano, F.: The Quest to Replace Passwords: A Framework for Comparative Evaluation of Web Authentication Schemes. In: Proceedings of the 2012 IEEE Symposium on Security and Privacy. SP '12, Washington, DC, USA, IEEE Computer Society (2012) 553–567
19. Kamath, S., Lindh, J.: Measuring Bluetooth Low Energy Power Consumption. Texas Instruments application note AN092, Dallas (2010)
20. Heydon, R.: Bluetooth Low Energy The Developer's Handbook. Prentice Hall (2013)
21. Sasse, M.A., Steves, M., Krol, K., Chisnell, D.: The great authentication fatigue–and how to overcome it. In: Cross-Cultural Design. Springer (2014) 228–239
22. Card, S.K., Moran, T.P., Newell, A.: The keystroke-level model for user performance time with interactive systems. Communications of the ACM **23**(7) (1980) 396–410
23. Laurie, B., Singer, A.: Choose the red pill and the blue pill: A position paper. In: Proceedings of the 2008 Workshop on New Security Paradigms. NSPW '08, New York, NY, USA, ACM (2008) 127–133
24. FIDO Alliance: FIDO U2F Spec Package (May 2015)
25. Want, R., Hopper, A., Falcao, V., Gibbons, J.: The active badge location system. ACM Transactions on Information Systems **10**(1) (January 1992) 91–102
26. Landwehr, C.E.: Protecting unattended computers without software. In: Proceedings of the 13th Annual Computer Security Applications Conference, Washington, DC, USA, IEEE Computer Society (December 1997) 274–283
27. Landwehr, C.E., Latham, D.L.: Secure identification system (1999) US Patent 5,892,901, filed 1997-06-10, granted 1999-04-06.
28. Peeters, R.: Security Architecture for Things That Think. PhD thesis, KU Leuven (June 2012)
29. Simoens, K., Peeters, R., Preneel, B.: Increased resilience in threshold cryptography: Sharing a secret with devices that cannot store shares. In Joye, M., Miyaji, A., Otsuka, A., eds.: Pairing 2010. Volume 6487 of LNCS., Springer (2010) 116–135
30. Cachin, C., Kursawe, K., Lysyanskaya, A., Strobl, R.: Asynchronous Verifiable Secret Sharing and Proactive Cryptosystems. In: 9th ACM Conference on Computer and Communications Security. (2002) 88–97
31. Tassa, T., Dyn, N.: Multipartite Secret Sharing by Bivariate Interpolation. In: 33rd International Colloquium on Automata, Languages and Programming. (2006) 288–299
32. Schultz, D., Liskov, B., Liskov, M.: MPSS: Mobile Proactive Secret Sharing. ACM Trans. Inf. Syst. Secur. **13**(4) (2010) 34:1–34:32
33. Texas Instruments: CC2541 SimpleLink Bluetooth Smart and Proprietary Wireless MCU. Web page

Appendix A Schema of the Pico credential database

Table 1 shows the schema of the Pico credential database. Each row in this database is indexed by a hash of the service's identifier $H(ID_S)$. Each of the N rows in the Pico credential database contains a 1-byte bin identifier $\beta_1, \ldots, \beta_N$, an encrypted credential, and other information required to identify the account to the user. Note that if $N > 256$, then there will be multiple credentials in the same bin; those credentials will be encrypted with the same key. A cold boot attack during a bin key reconstruction will reveal all of the credentials in the bin, and not just the credential being actively requested—but only a fraction of one percent of all the credentials, and not all of them, as in the previous design.

Table 1. Schema of the Pico credential database. Depending on the scenario, credentials may be public/private key pairs or symmetric keys shared with the service. Note that Google and Expedia end up in the same bin, that the user shares the same userid for Expedia and Amazon, and that the user has two distinct Twitter accounts, which she distinguishes by the userid.

Hash of service's identifier	Bin identifier	Encrypted credential	Userid
$H(ID_{Google})$	0x1e	$\{cred_{Google,jane.doe}\}_{K(0x1e)}$	jane.doe
$H(ID_{Amazon})$	0x75	$\{cred_{Amazon,jane257}\}_{K(0x75)}$	jane257
$H(ID_{Twitter})$	0x57	$\{cred_{Twitter,@jane}\}_{K(0x57)}$	@jane
...	...	...	...
$H(ID_{Expedia})$	0x1e	$\{cred_{Expedia,jane257}\}_{K(0x1e)}$	jane257
$H(ID_{Twitter})$	0x32	$\{cred_{Twitter,@tattoophile}\}_{K(0x32)}$	@tattoophile

Appendix B Hardware Prototype Platform

Our prototype Picosiblings have been developed using an off-the-shelf BLE development kit from Texas Instruments (TI), shown in Figure 3. The TI development board is built around a power-optimised system-on-chip (SoC) solution targeting BLE and proprietary 2.4 GHz RF applications [33]. The development kit includes a hardware debugger and full source code for Operating System (OSAL) and Hardware Abstraction Layers (HAL). The BLE protocol stack is provided as a set of binary libraries. Software for the platform is built using the third-party IAR Embedded Workbench toolchain.

The CC2541's main features include:

- High-performance low-power 8-bit 8051 processor
- 256 KB flash and 8 KB RAM (retained in all power states)
- Peripherals including watchdog and general-purpose timers, 2x USART, I2C, and AES coprocessor
- 6 mm x 6 mm QFN40 package

Fig. 3. Texas Instruments CC2541DK-MINI Bluetooth Low Energy development kit.

Fig. 4. The two standard coin batteries we used. Left: CR1616 (16 mm diameter, 1.6 mm thickness, ≈55 mA h). Right: CR2032 (20 mm diameter, 3.2 mm thickness, ≈230 mA h).

In volume, the CC2541 SoC is priced at approximately $4. However, assuming price falls in line with trends witnessed with classic Bluetooth, it can be expected to approach the $2–3 range as the technology becomes ubiquitous. As the CC2541 SoC requires relatively few additional discrete components the bill of materials for a Picosibling built using this device could comfortably sit in the $3–4 range. This is, arguably, sufficiently cheap for users not to worry about purchasing and replacing devices.

Standard button-cell CR1616 and CR2032 batteries (shown in Figure 4) are reasonable batteries to use in our scenario, as their size makes them well suited to wearable electronics and embedding in jewellery.