Am I in good company? A privacy-protecting protocol for cooperating ubiquitous computing devices

Oliver Stannard and Frank Stajano

University of Cambridge Computer Laboratory
15 JJ Thomson Avenue, Cambridge, CB3 0FD, United Kingdom

Abstract. A portable device carries important secrets in encrypted form; to unlock it, a threshold secret sharing scheme is used, requiring the presence of several other devices. We explore the design space for the protocol through which these devices communicate wirelessly, under the additional constraint that eavesdroppers should not be able to recognize and track the user carrying these devices.

1 The problem

The user carries a device, the Pico [11], containing important secrets (authentication credentials for all of the user's accounts) that must not be disclosed to attackers. The memory of the Pico is encrypted with a key that the Pico periodically forgets and reconstructs, through k-out-of-n secret sharing [9], from the shares it receives via short-range radio from other devices, the Picosiblings. The Picosiblings are meant to be physically bound to the user more tightly than the Pico itself, so that the Pico can equate the proximity of the Picosiblings with the proximity of the Pico's owner.

Assuming that the secret has already been split into n shares and that the devices have already been suitably initialized, what protocol should they use to communicate? The protocol must offer the following properties (from the original Pico paper [11]):

- The Pico can ascertain the presence of any of its Picosiblings in the vicinity.
- Each Picosibling responds to its master Pico but not to any other Pico.
- At each ping, each 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 [3]).

Our attacker model is essentially Dolev-Yao [1] as far as messages go (the adversary can overhear, intercept and modify all traffic) but with the added twist that the attacker may also capture the Pico (while it is in a locked state) and some of the Picosiblings (fewer than k). We assume the adversary *cannot* capture the Pico while it is unlocked and that the adversary *cannot* capture more than k Picosiblings.

The main goal of the attacker is to unlock the Pico: if this happens, it's game over. Another goal of the attacker is to recognize an individual user from the overheard messages between Pico and Picosiblings: since the Pico and the Picosiblings are strongly tied to an individual, this ability might be used to violate the individual's location privacy.

We believe the problem is hard enough that we don't want to optimize prematurely: we won't add extra constraints before we have found at least one correct solution. Once we have one, though, additional nice-to-have properties would take into account the physical constraints on the ubicomp devices, in particular the very limited energy available to each of the Picosiblings[1].

1.1 Timing and storage details for the Pico

This subsection gives some background about the lifetime of the shares and of the shared secret but is not essential to understand the protocol between Pico and Picosiblings.

The high level protection goal is that, when the Pico is away from its Picosiblings, it becomes locked in a way that prevents an attacker from retrieving the authentication credentials, even if the attacker can totally dismantle the Pico and read every bit stored in it. This is achieved by encrypting all[2] the non-volatile memory of the Pico with a master key that is securely erased when the Pico is locked.

Since the protocol described in this paper (communication between Pico and Picosiblings) is what allows a locked Pico to reconstruct its master key, it must obviously be allowed to run even while the Pico is locked. Therefore, if it needs access to any persistent state, that state cannot be part of the memory that gets encrypted under the master key; that state will instead be available to an attacker who captures a locked Pico.

We assume that the hardware architecture of the Pico ensures that, if the attacker captures an unlocked Pico, it is not feasible to extract the master key or the shares from the Pico before they are securely erased. Discussion of how to achieve that is outside the scope of the present paper but we envisage using a Hardware Security Module[3] or perhaps techniques similar to those pioneered by TRESOR [6], whereby keys are only ever stored in processor registers rather than in RAM. We may also wish to encrypt the RAM itself, as suggested in various cryptoprocessor designs such as Kuhn's TrustNo1 [5].

We define system parameter T_1 as the maximum time interval that the Pico will remain unlocked once it is no longer in range of at least k Picosiblings. We

[1] In an ideal world they'd be so parsimonious with energy that they could be implemented as passive RFID tags.

[2] Well, almost all—see next paragraph.

[3] Such as the TPM (trusted platform module) chip featured in many modern laptops.

expect it to be in the range between a second and a minute: the exact value will have to be tuned with user testing. It needs to be short for security reasons, but not too short for usability and power consumption[4] reasons.

The Pico requests a share from each of its n possible Picosiblings, staggering the requests so that each Picosibling is contacted in succession at a definite time depending on the Picosibling's number[5]. The cycle time is set so that the interval between successive requests for the same share is shorter than T_1 by some margin.

Once share s_i is received, the Pico loads the share's countdown timer[6] with the value T_1. If the shared secret is not there and at least k shares have a non-expired timer, the shared secret is recomputed. If the same share is received again before the countdown timer expires, the timer is reloaded to T_1; but when the timer eventually expires, the share is securely wiped and the shared secret is recomputed using the remaining available shares, or securely wiped if fewer than k shares remain.

2 A first attempt

With disregard for computational costs and power consumption concerns, we first propose a solution based on public key cryptography.

The protocol between the Pico and each of its Picosiblings consists of a ping from the Pico and a pong from the Picosibling.

The Pico has a key pair for each of its Picosiblings: $k_{1,j}, k_{1,j}^{-1}$, with j ranging over the Picosiblings. Each Picosibling too has a key pair: for Picosibling number j it will be $k_{2,j}, k_{2,j}^{-1}$.

The ping message, from Pico to Picosibling, means: "hey, Picosibling j, are you there? If yes, please send me your key share s_j." It consists of a random nonce r, signed by the Pico with its private key $k_{1,j}^{-1}$ so that the Picosibling can verify its provenance and encrypted under the Picosibling's public key[7].

$$\text{Ping for Picosibling } j, \text{broadcast by Pico}: \quad E_{k_{2,j}}(S_{k_{1,j}^{-1}}(\texttt{"ping"}, r))$$

[4] Shortening T_1 increases the minimum frequency of pings from Pico to Picosiblings and therefore increases the radio communication costs.

[5] This allows the Picosibling to go to sleep most of the time except when it expects a transmission from the Pico. This assumes that the Picosibling has a continuously-running clock, and that running the clock has negliglible cost compared to turning on the radio for listening. We need to address the issue of clock sync between the Pico and the Picosibling, especially in case we also allow the Picosibling to be switched off completely, clock included.

[6] This is only a conceptual description—the implementation does not need to keep a separate countdown timer for each share so long as the effect is the same.

[7] This protocol plays it safe but is potentially redundant; while we do want to protect the integrity and source authenticity of the nonce r, it is not clear that we need to protect its confidentiality.

Note that the so-called "public" key $k_{1,j}$ isn't actually public: it is only known to the Pico and to Picosibling j, so nobody else can verify that signature.

On receiving a ping message, Picosibling j verifies the signature[8] using the Pico's "public" key $k_{1,j}$. If the signature matches, the Picosibling proceeds. If it does not match, the Picosibling ignores this message.

Having recognized a valid signature, Picosibling j composes a pong message by concatenating the received nonce r with its share s_j, signing the lot with its private key $k_{2,j}^{-1}$ (so that the Pico can verify its provenance) and then encrypting the signed message under the Pico's "public" key $k_{1,j}$.

$$\text{Pong from Picosibling } j: \quad E_{k_{1,j}}(S_{k_{2,j}^{-1}}(\text{"}\texttt{pong}\text{"}, r, s_j))$$

Here the encryption is necessary, to prevent eavesdroppers from learning s_j, but it could be argued that the signature is redundant, given that no other principal is supposed to know the Pico's "public" key $k_{1,j}$. If we explicitly treat this "public" key as a secret known only to the Pico and to Picosibling j, then the $k_{2,j}, k_{2,j}^{-1}$ key pair of the Picosibling is no longer necessary: the Pico knows that the message came from Picosibling j because no-one else could have encrypted it under $k_{1,j}$.

At this point one wonders why $k_{1,j}$, if it's shared only between the Pico and Picosibling j, should be a "public" key instead of a symmetric key. More on that in section 3, but meanwhile here is a major vulnerability that already kills this first attempt.

2.1 Attack

Assume the attacker records the traffic between the Pico and a sufficient number of Picosiblings. Later, the attacker captures the locked Pico. Having extracted $k_{1,j}^{-1}$, which by hypothesis is unencrypted even in the locked Pico[9], the attacker decrypts a previously recorded pong from Picosibling j and recovers s_j. Provided that the attacker could record the traffic with at least k Picosiblings, he or she now has at least k shares and can recover the master secret that unlocks the Pico. Game over.

3 A symmetric-key ping-pong protocol

We switch to symmetric keys, not to save energy (though that's always nice) but primarily to be able to renew keys in order to defeat the previous attack.

[8] Clearly there is the possibility of what Stajano and Anderson [12] called a "sleep deprivation torture" (battery exhaustion) attack against the Picosibling, by forcing the Picosibling to verify bogus signatures. Under our stated assumptions we won't worry about it until we have a solution to the main problem.

[9] Otherwise the locked Pico could not run this protocol to obtain the shares that allow it to unlock itself. Thus the $k_{1,j}^{-1}$ keys, for all j, are only protected by whatever modest tamper resistance we can add to the Pico, so we must assume that an attacker with unsupervised physical access to the Pico will be able to retrieve them.

Now the Pico shares a symmetric key $k_{3,j}$ with Picosibling j, which is used for Encrypt-then-MAC in both the ping and the pong. However, after every ping-pong, the two parties renew this symmetric key by hashing it: this stops the previous attack because the key that the attacker finds in the Pico cannot be used to read past messages. A counter c_j is included next to the random nonce r so that the two parties can stay in sync with the number of hashes of the key even if some messages get lost in transit.

The protocol still consists of only two messages, a ping and a pong, followed by a local hash computation at both ends:

$$\text{Ping for Picosibling } j, \text{broadcast by Pico}: \quad \mathit{EtMAC}_{k_{3,j}}(\text{"}\texttt{ping}\text{"}, r, c_j)$$

$$\text{Pong from Picosibling } j: \quad \mathit{EtMAC}_{k_{3,j}}(\text{"}\texttt{pong}\text{"}, r, c_j, s_j)$$

$$\text{Both (with appropriate sync)}: k_{3,j} := h(k_{3,j})$$

Why Encrypt-then-MAC [4] rather than just MAC in the ping and just Encrypt in the pong? As with the previous protocol, we wish to preserve the integrity of the nonce r, which justifies the MAC in the ping. But this time we also have a good reason for *encrypting* the ping: we want to hide the counter c_j from the eavesdropper, who might otherwise use it to correlate pings from the same Pico to mount a privacy attack. As for the pong, we wish to protect both the confidentiality and the integrity of the share s_j.

3.1 Attack

Although we stopped the previous attack, a more elaborate variation of it still applies, although it is a little harder to mount for the attacker.

The attacker steals the locked Pico, extracts all its keys $k_{3,j}$ and counters c_j, extracts a copy of the encrypted non-volatile memory, then *returns* the Pico and hopes the victim will continue to use it without noticing the tampering. If so, the attacker records the future traffic between Pico and k different Picosiblings, decrypts it using the extracted keys and recovers k shares and hence the master key that decrypts the extracted memory image. Game over again. With this countermeasure we can stop the attack towards the past but not the one towards the future.

We can't even tell the user: "if you lose your Pico, redo all your secret sharing before using it again", because the user can't easily tell whether he "lost" the Pico. If he just left it in his room *and found it there when he returned*, how does he know that the attacker didn't come and image it and then left it there? This can be seen as a distant relative (same social setting, different technical steps) of Rutkowska's "evil maid" attack [8].

It would appear that a sensible countermeasure against this attack is to make the Pico tamper-evident[10], although security then dangerously depends on the user always inspecting the seal with sufficient care.

[10] Which is much less demanding than tamper-resistant, and thus not unreasonable.

4 Reset

A related design issue came up while implementing simulations of these protocols: when and how can a user reset the Pico or the Picosiblings to their factory default state? Can the attacker do the same? If yes, is there any advantage in doing so?

In the context of the general problem of resetting unattended devices to factory default, we [10] previously distinguished between the simple but effective Big Stick Principle ("anyone with physical access to the device is allowed to take it over") and the more elaborate Resurrecting Duckling security policy model, in which the mother duck can order the duckling to commit seppuku[11] but where the duckling device must be sufficiently tamper resistant to prevent a third party from doing the same.

In the Pico paper we said the Pico can be reset by invoking seppuku from the unlocked state. So this is a case of Resurrecting Duckling in which the Pico is the duckling and the mother duck is any cloud of k or more Picosiblings[12]. Therefore, only someone who holds enough Picosiblings can reset the Pico.

How about resetting a Picosibling itself to factory default? We don't want an infinite regression of duckling pairings, so we make the Picosiblings obey the Big Stick Principle. We could have a physical reset button on the Picosibling, for example, which would make it forget its association with its Pico. The Picosibling is thus subject to denial of service (adversary pressing the reset button and cancelling the security association, causing the Picosibling to stop working as intended), but it is anyway because someone with physical control over it could equally easily just smash it to pieces. Besides, if the adversary has physical control of the Picosibling, what worries us much more than denial of service is the possibility that, with enough of them, he might extract the share (or just use the Picosibling as is) to unlock the Pico.

5 Re-pairing attack

Another attack, which is independent of the ping-pong protocol, consists of stealing the Pico from the victim while it is still unlocked (grab-and-run style theft) and then, before T_1 expires, re-pairing k new (blank) Picosiblings to the stolen Pico. Then the attacker can keep the Pico unlocked indefinitely and extract all its secrets.

This attack is, technically, outside our attacker model, because the Pico is captured while unlocked; however it is also relatively easy to prevent.

One possible countermeasure is to ensure that the process of adding a new Picosibling takes longer than T_1, so the operation can't be completed without being in the aura of k already-paired Picosiblings to keep the Pico unlocked.

[11] Seppuku is the ritual suicide of the samurai. In this context it means, essentially, that the duckling will reset itself, forgetting any existing security associations and preparing to be taken over by a new master.

[12] We gloss over the business of the additional special shares like the biometric and the remote server [11].

Another possible countermeasure is to ensure that changing the set of Picosiblings requires the two special shares (where the biometric share in particular is used to indicate explicit user consent).

The two countermeasures could even be combined, for good measure.

6 Related work

The most comprehensive work about this problem space is Peeters's PhD dissertation [7], published a couple of months after this workshop, which addresses related issues in much greater depth and provides much more elaborate, robust and mathematically verified solutions than the protocol we sketched here. On the security side, it offers a method for storing shares on devices (like the Picosiblings) that do not offer secure rewritable storage; the main requirement is that the devices be able to store, in tamper-proof secure storage (such as a PUF [2]), an immutable factory-defined private key. On the privacy side, it offers a privacy-protecting mutual authentication protocol, efficient enough to be usable with RFID tags and capable of supporting multiple readers. Unfortunately, though, it does not appear easy to combine these two useful but independent primitives into a single solution that would offer the benefits of both.

References

1. Danny Dolev and Andrew Chi-Chih Yao. "On the security of public key protocols". *IEEE Transactions on Information Theory*, **29**(2):198–207, 1983.
2. Blaise Gassend, Dwaine Clarke, Marten van Dijk and Srinivas Devadas. "Controlled Physical Random Functions". In "ACSAC '02: Proceedings of the 18th Annual Computer Security Applications Conference", p. 149. IEEE Computer Society, Washington, DC, USA, 2002. ISBN 0-7695-1828-1.
3. Gerhard P. Hancke and Markus G. Kuhn. "An RFID Distance Bounding Protocol". In "Proc. IEEE SECURECOMM 2005", pp. 67–73. 2005. `http://www.cl.cam.ac.uk/~mgk25/sc2005-distance.pdf`.
4. Hugo Krawczyk. "The Order of Encryption and Authentication for Protecting Communications (or: How Secure Is SSL?)". In Joe Kilian (ed.), "CRYPTO", vol. 2139 of *LNCS*, pp. 310–331. Springer, 2001.
5. Markus Kuhn. "The TrustNo 1 Cryptoprocessor Concept", 1997. `http://www.cl.cam.ac.uk/~mgk25/trustno1.pdf`. Manuscript.
6. Tilo Müller, Felix C. Freiling and Andreas Dewald. "TRESOR Runs Encryption Securely Outside RAM". In "USENIX Security Symposium", USENIX Association, 2011. `http://www.usenix.org/event/sec11/tech/full_papers/Muller.pdf`.
7. Roel Peeters. *Security Architecture for Things That Think*. Ph.D. thesis, KU Leuven, Jun 2012. `http://www.cosic.esat.kuleuven.be/publications/thesis-202.pdf`.
8. Joanna Rutkowska. "Why do I miss Microsoft BitLocker?", 2009. `http://theinvisiblethings.blogspot.co.uk/2009/01/why-do-i-miss-microsoft-bitlocker.html`.
9. Adi Shamir. "How to Share a Secret". *Communications of the ACM*, **22**(11):612–613, Nov 1979. `http://securespeech.cs.cmu.edu/reports/shamirturing.pdf`.

10. Frank Stajano. *Security for Ubiquitous Computing.* Wiley, 2002. ISBN 0-470-84493-0.
11. Frank Stajano. "Pico: No more passwords!" In Bruce Christianson et al. (ed.), "Proc. Security Protocols Workshop 2011", vol. 7114 of *LNCS*. Springer, Mar 2011. `http://www.cl.cam.ac.uk/~fms27/papers/2011-Stajano-pico.pdf`.
12. Frank Stajano and Ross Anderson. "The Resurrecting Duckling: Security Issues in Ad-Hoc Wireless Networks". In B. Christianson et al. (ed.), "Proc. Security Protocols Workshop 1999", vol. 1796 of *LNCS*, pp. 172–182. Springer, Apr 1999. ISBN 3-540-67381-4. `http://www.cl.cam.ac.uk/~fms27/papers/1999-StajanoAnd-duckling.pdf`.