A Secure Framework for OTA Smart Device Ecosystems Using ECC Encryption and Biometrics

Miguel Salas

Intel Corporation, Microprocessor Group, Fort Collins, Colorado, USA
miguel.o.salas@intel.com

Abstract. As we move towards a world where all the traditional household appliances and basic industrial devices are being transformed into interactive high-computing devices, an ecosystem of these smart devices is emerging. With this impending revolution, often coined the *Internet of Things*, one of the understated challenges is the security infrastructure that must accompany the deployment of this ecosystem. In this paper we propose a security framework that leverages hierarchical hardware memory mapping, modularity of the Operating System, and an efficient biometric aided ECC cryptosystem to work together towards this security need. We focus on the secure and efficient implementation of OTA updates and inter-device communication. Our work shows that by integrating several novel improvements based on real system considerations with state-of-the art techniques, we can build a commercially feasible security framework for these devices that is 35% faster and 5% more load efficient than current state-of-the-art ECC-based cryptosystems and OTA compression schemes.

Keywords: OTA, security framework, biometrics, elliptic curve cryptography, Internet of Things.

1 Introduction

While the use of smartphones is already ubiquitous and the number of software-controlled electronic components in a car continues to increase at a fast rate, we have yet to see a widespread deployment of other devices capable of smart computation and high user interaction. These devices range from printers and TVs, whose smart versions are slowly being introduced to the market, to consumer electronics, such as refrigerators and thermostats whose smart versions are yet to achieve a significant consumer base.

While these smart devices lack strong connectivity among each other, their design is trending towards high performance computing with innovative capabilities and high interconnectivity among them. For example, envision a smart shower system, a thermostat with access to the outside weather, and a smart car, all Wi-Fi enabled. If the smart shower system learns that the user will want to drive his car within 10 minutes of getting out of the shower, it can query the thermostat to determine whether the weather outside is below a certain temperature, it will request the car engine to start as soon as the user is out of the shower. Similarly, a smart fridge can keep track of all

A.I. Awad, A.E. Hassanien, and K. Baba (Eds.): SecNet 2013, CCIS 381, pp. 204–218, 2013.

items stored inside, and if a grocery item is running low, it can query the closest laptop in the house to place an order on an online grocery-shopping site. In this manner, these devices can create a *smart device ecosystem*. Figure 1 illustrates an ecosystem of this nature.

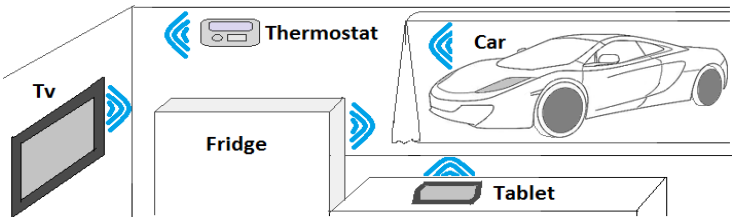

Fig. 1. Sample ecosystem of smart devices of the future

Connectivity enablement for these ecosystems has already been recognized as a need [1], and has been the leading motivation to create Operating Systems such as Tizen [2], which can enable a web API based connectivity infrastructure among all these devices. However, while providing a natural next step towards smart device evolution, these ecosystems make attractive targets for security breaches. Going back to the examples above, a malicious attacker can now masquerade as the shower system, creating a replay attack to turn on the car engine whenever he desires, or masquerade as the fridge to charge any grocery items it pleases to the user's credit card. Hence, security for such ecosystems is imperative.

In this work, we propose a framework to address this security need. Unlike the previously proposed security frameworks, our work leverages three key unique aspects of these devices: (1) Unlike nodes on a WSN, these devices are expected to have some level of user interaction; (2) Unlike mobile phones, because these devices live within a geographically-limited network, we must consider efficient intra-net communications among them; (3) Given the known software and hardware deployment patterns of these devices, there are optimization areas at the software-hardware integration level. Our work proposes a strong, efficient, and flexible security framework, which takes into account unique aspects of smart devices including their physical memory layout, the patterns of OTA update deployments, and the need for ergonomic and cost-efficient designs.

The rest of the paper is divided into seven sections. Section 2 discusses previous work on the different areas that make up security infrastructures of this nature and Section 3 discusses the high level view of the proposed infrastructure. Section 4 contains the description of the framework, including the hierarchical OS design, OTA transmission protocol, proposed encryption algorithms and the biometric sensor integration. Section 5 presents the system performance results and Section 6 concludes the paper.

2 Related Work

The imminent arrival of the Internet of Things (IoT) infrastructure is best envisioned in [1]. Any IoT device must possess an OTA update capability. Several OTA-enabled frameworks have been proposed. The essential OTA enablement steps are outlined in

[3]. The OTA-PSD framework [4] shows the underlying mechanism from the user and provider necessary for OTA updates, including the XML based software versioning scheme. The OTA-WNS framework [5] shows how to model a device's state as an OTA-update-centric state machine.

Furthermore, any OTA framework needs security, as outlined in [6]. The security extension for OTA-enabled devices proposed in [7] relies on the use of the TLS extension over TCP packets. Similarly, the SenseOP framework [8] makes use of ECC encryption to provide security guarantees assuming tamper proof device interfaces, which do not always hold. In [9], a hash chains based security framework for OTA updates on smart cars is proposed, although it is computationally light, it relies exclusively on tamper proof devices that make use of private keys. Similar to the SenseOP protocol, such assumptions will create single point vulnerabilities in IoT infrastructures. Our framework does not rely on these assumptions.

An OTA-enabled IoT device also needs to take into account the process of selecting the data for the OTA update. Since an OTA update must use network bandwidth, we are concerned with minimizing the transmission data and latency required. Recently proposed efficient solutions such as The Two-step Differential [10] and Queen Differential [11] are schemes that transmit only the difference between the current and new software and firmware version. Regardless whether such differential schemes are applied, typically the data is compressed. Conventional compression schemes are not ideal in this model, since the computing should be loaded onto the server whenever possible. In contrast, Byte Pair Encoding, explored in [12] and [13], is an asymmetric technique fit for this scenario.

Our framework relies on an improved version of Queen Differential, QDiff. The raw QDiff scheme relies in the two-stage differential process shown in figure 2 [10]. While QDiff can achieve small deltas in the image, it does so with some drawbacks. First, its space pre-allocation creates some slop spaces when existing code is deleted. This has the side effect of fragmentation over the long term. Furthermore, once it has found the delta it sends the delta over the network without specifying a compression scheme. Moreover since QDiff algorithm runs in exponential time with the size of the code in question, a large code database can make its runtime prohibitively large. Finally, QDiff is not optimized towards exploiting the lifecycle patterns of the OTA updates, which will be discussed in section V.

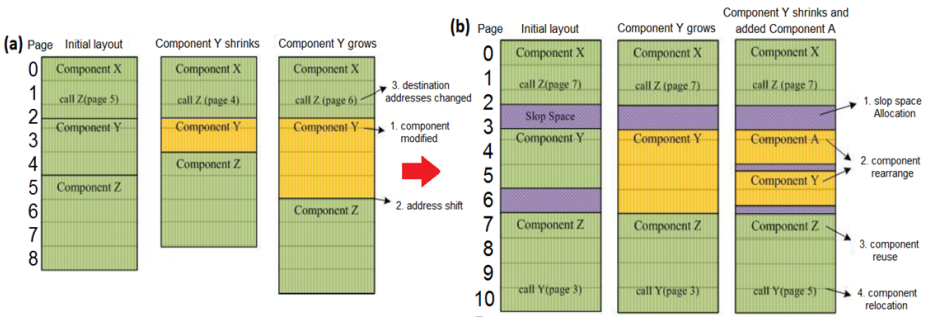

Fig. 2. Sample addition/deletion of new code for small-delta OTA updates using two-stage differential schemes such as QDiff. (a) Shows an update when a memory component (Y) grows. (b) How the scheme attempts to minimize component delta and slop space.

Finally, our framework relies on public cryptography because the devices are inherently heterogeneous and hence do not share a secure channel. Among public cryptosystems, the RSA public cryptosystem is the most widespread protocol, relying on the difficulty of factoring large numbers and finding discrete logarithms. According to the NISA, we can achieve a similar security offered by RSA cryptographic systems with Elliptic Curve Cryptography (ECC) through the use of a much smaller key size. A comparison of key sizes is shown in table 1 [14]. The performance advantages of ECC over RSA are outlined in [15], and security frameworks for mobile devices based on ECC are proposed in [16] and [17]. Note that as specified by the NISA, this relies on choosing a strong ECC curve.

Table 1. Equivalent security strength guarantees between symmetric key algorithms, RSA and ECC, by key sizes

Symmetric Key Algorithms	RSA	ECC
80	$k = 1024$	$f = 160\text{-}223$
112	$k = 2048$	$f = 224\text{-}255$
128	$k = 3072$	$f = 256\text{-}383$
192	$k = 7680$	$f = 384\text{-}511$
256	$k = 15360$	$f = 512+$

Finally, the key to our scheme is the use of biometrics. However, the use of biometrics to authenticate users typically suffers from accuracy issues. Proposed solutions can be divided into multi-sensor devices [18][19][20], whereby two or more biometric sensors are taken as inputs to improve accuracy, or multi-algorithm, whereby two or more algorithms are run independently based on the same biometric sensor. The former approach suffers from poor ergonomics: asking a user to first log in with a face read, then with a fingerprint, and finally with a retina reading might yield a high-accuracy, high-security system, but it would be far from practical for consumer devices. The latter approach suffers from the limitations of the biometric sensor itself. If a biometric sensor reads a fingerprint and takes several patterns of it, the finger itself can become a single point of failure. Instead, our approach is a simple and ergonomic solution based on a hybrid multi-sensor approach.

3 High Level Overview

In the smart device ecosystem, there are three forms of communications: (1) from the device to the OTA provider, (2) among the smart devices inside the ecosystem in an ad-hoc wireless network, and (3) from the devices to other HTTP servers on the internet. This is shown in figure 3(a). In this paper we are exclusively concerned with (1) and (2) because it is assumed that (3) can be handled using HTTPS requests and depends uniquely on the HTTP server policies of which we have no control over. Thus in our framework we optimize towards those two communication paths. Furthermore we model communication between the manufacturer and the device to be exclusively OTA updates. We assume that the hardware manufacturer and the operating system provider act as a joint entity, henceforth referred to as the manufacturer, for device updates. Therefore, we assume that OTA updates concern all of the ROM contents: the operating system, the firmware, and immutable native user applications. Therefore, in this paper, any OTA update reference is interchangeable with OTA-ROM updates. The high-level scheme is depicted in figure 3(b).

Our contributions are as follows. We exploit hierarchical memory mapping for OTA benefits and encrypting a secure key. Furthermore, we propose leveraging the software update versioning scheme in use today for OTA updates. Finally, we propose a novel fingerprint-based biometrics to provide user authentication and aid the ECC encryption protocol.

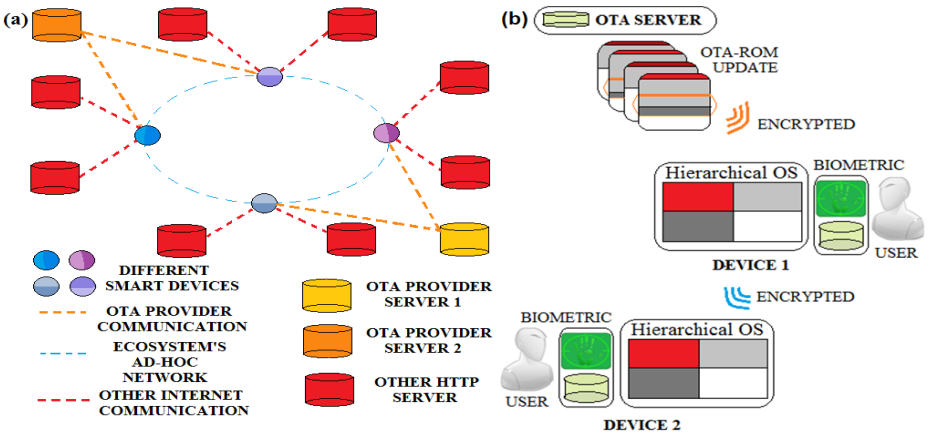

Fig. 3. (a). High level overview of the framework. The three types of communication in the smart device ecosystem: (1) from the device to its OTA server, (2) intra communication, and (3) from the device to other HTTP servers. Notice that we assume heterogeneous devices and providers. **(b)** High level model of the proposed security framework focusing on (1) and (2)

4 Security Framework Description

Our framework consists of four different components: a hierarchical memory mapping for OTA updatable components, an OTA protocol, an encryption scheme, and a biometric sensor support. Each of these is described in detail below.

4.1 Operating System Support

We propose the use of a modular Operating System and its firmware components so that hierarchical memory mapping in the ROM is achievable. We propose dividing up

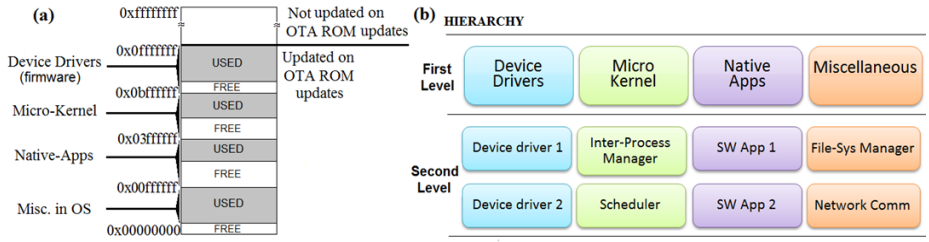

Fig. 4. Memory organization for the OTA components **(a)** Shows the proposed memory partition with sample memory addresses. **(b)** Shows a two level hierarchy that although one could use to partition the memory would result in severe fragmentation.

the ROM into a four component hierarchy, so that ROM updates can be treated separately for each of these four hierarchies. Figure 4(a) shows the proposed division.

Note that we arbitrarily choose three major hierarchies and allocate a fourth hierarchy for the rest. Although we could extend this to a two-level hierarchy as explained conceptually in figure 4(b), we do not do so because more partitions are likely to cause code fragmentation and reduce the ability of the compiler to allocate memory space efficiently.

4.2 OTA Update Protocol

The trend for smart device development, from game consoles to smartphones, is to release software and firmware OTA updates backwards compatible with the devices up to a target device life cycle. For example, Android and iOS devices can continue receiving firmware updates from the device manufacturer up to a few years out in time, after which the device is no longer supported. The target device life varies sharply among devices, with gaming consoles lasting roughly a decade. However, software and/or firmware OTA updates follow a cyclical pattern such that there is a major upgrade, encompassing new device features, followed by several minor firmware updates, typically targeted towards bug fixing. Figure 5 illustrates the typical firmware OTA update deployment cycle. As mentioned in section 2, in our framework, we propose using a modified version of the Queen Differential (QDiff): the *Modular and Cycle Aware extended Queen Diff* (MCA-QDiff).

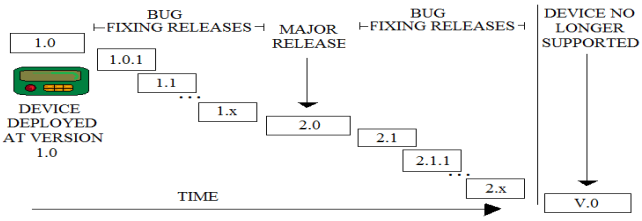

Fig. 5. Typical firmware update lifecycle for smart devices, including minor and major releases

The modularity of the OS memory mapping enables MCA-QDiff to operate at a finer grain than the original QDiff by enacting the code delta on a ROM hierarchy only if it was marked as touched. To obtain this functionality, at the time that an OTA is deployed, a header packet contains information about whether a ROM hierarchy will receive an update or not. Furthermore, because the order of the ROM is hierarchical as described in section 4.1, the area of the ROM that each of these hierarchies occupy is bounded by address ranges. This allows the code delta to be applied only at that level of hierarchy. For our proposed infrastructure, there are four hierarchies living in the OTA updatable ROM image: micro-kernel, native software applications, device drivers, and miscellaneous functions that do not fall into any of the previous three categories. Inside any of these hierarchies, memory boundaries blur and we let the compiler produce an efficient image implementation. Once a hierarchy in the ROM has been marked as needing update by the header packet, it applies the raw QDiff approach.

As mentioned in section 2, QDiff adds jump instructions to link the current code with the new code and pads where the old code is deleted, thus minimizing compile image differences. However, conducting this procedure repeatedly can create excessive padding in the gaps of the erased code leading to code fragmentation, and is especially inefficient when the code deltas are very large. In fact, there is nothing stopping the raw QDiff to produce an image delta that is larger than the whole image itself. However, because we now have an idea of how the ROM updates are being performed, we know that it is not reasonable to send a compressed image when it has had major code overhaul. So instead, MCA-QDiff chooses to send the complete image of a given hierarchy when such overhauls happen, and compute the ROM code delta only on minor updates. Note that even on major updates, the full image is not sent. For example, if the Micro-kernel and device drivers stayed the same, yet all else changed, it would send a header stating so, along with the changes for the other two chosen hierarchies.

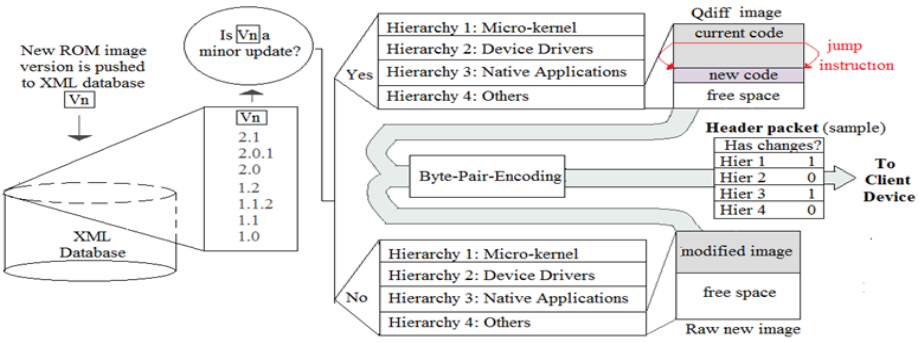

Fig. 6. Proposed MCA-QDiff transmission protocol for OTA ROM image updates.

Whether it was a minor update with a minimal image delta produced or a major update with the selectively sent full image, it is compressed prior to sending using Byte Pair Encoding as discussed in section 2. This yields a small image for the minor firmware version releases, which is asymmetrical in computing time: it takes much longer to encode at the server than to decode inside the device, meeting our design goals. Finally, we leverage the XML based approach to keep track of the ROM image versioning [4]. The entire OTA transmission protocol is shown in figure 6.

4.3 Encryption Protocol

Due to the high number of opportunities for malicious attacks, encrypted communications are a must in this ecosystem. Previously proposed approaches have made use of the fact that one can embed a secret key on the hardware ROM itself, often leveraging the serial number in the device and storing it in the manufacturer's server database. This approach has two major drawbacks: (1) all encrypted device communications must be between the device and the manufacturer's server that possesses this key, and (2) anyone able to compromise the server, can now masquerade as the server to access information on the target device. Thus, we turn to public key encryption to bypass

these challenges at the cost of increased overhead in the communications. We argue, however, that since this overhead is only paid when establishing new communication sessions, it is tolerable, given its higher flexibility, resilience, and security. Furthermore, as explained in section 2, we choose public cryptography based on ECC over RSA due to its smaller key size. ECC is also a computational asymmetric protocol: calculating a computing intensive elliptic curve can be done on the server side, while the rest can be done on the client side. Hence, we rely on ECC for encryption.

To improve performance, our framework attempts to leverage the secret key channel established by the manufacturing stage process. In our framework, the manufacturer will install a set of private keys in the device's ROM memory. We choose a 224 bit ECC system because it is equivalent to the 2048 bit key RSA. Notice that a 224 bit key requires only 7 32-bit registers in RAM for operation, while a 2048 bit key RSA requires 64 32-bit registers, which is roughly the size of an entire register set found in simple low-power microprocessor architectures. Thus, calculations involving RSA keys would slow down hardware performance significantly.

Private Key Protection. A public key cryptosystem with a compromised private key is no longer reliable. For this reason, our framework takes into account the two main security risks in private key deployment: (1) attempting to compromise the ROM location where the keys are stored with a bit flip, and (2) attempting to extract the key from the ROM location without leaving a trace of device tampering. Note that the former also covers reliability, since alpha particles or manufacturing defects can lead to bit flips.

To address the bit flip concerns, we create a redundant copy of the key in the ROM at manufacturing time. Because we are trying to address bit flips, we cannot store the key copies in adjacent memory locations, due to the locality principle of manufacturing defects and typical ROM attacks. However, we cannot place them at random memory locations as this would create unnecessary code fragmentation. Therefore we leverage the memory hierarchy proposed in section 4.1 by placing the redundant keys at the start of each of these hierarchies. To address device tampering on the secret key, the manufacturer will not just install a key, but instead create two 112 bit keys which will be concatenated to create the 224 bit key. Because we assume keys are stored in pages with 32-bit addressing, the device simply needs 4 memory locations for each of the two 122 bit keys, with the last memory location per key padded with 16 bits of random data. Such system avoids an unnecessary XOR or more complex hash operation to produce a key based on two other keys at run-time. We assume that while a malicious user might snatch one of the 122 bits private keys, it is unfeasible that he might find both and know to concatenate them.

Finally, since we need a dual key system for the reliability concern and a dual key system for the tampering concern, we use a set of four 122-bit keys, which match seamlessly with the four hierarchies used in the memory mapping. The full private key installing scheme, which we refer to as the Memory Hierarchy based Private Key (MHPK) scheme is illustrated in figure 7. Note that our system can detect bit flips and key tampering because after the first set of keys do not yield successful authentication, it will attempt its second set of keys. In the extremely rare case that the second key also fails, to avoid leaving the device in an obsolete state, we rely on biometric user authentication to request a ROM update from the server, as will be discussed in section 4.4.

Session Key Generation. While having a reliable private key in the device is the first step towards an ECC cryptosystem, it is far from enough. In our framework, the manufacturer will install the device with its public key as well. This requires calculating the Elliptic Curve math at install time. The server will calculate a strong elliptic curve, pick a point on the curve and come up with the device's public key based on the curve parameters. Once the device public key has been determined, the manufacturer will install the device's public key and the manufacturer's public key in the ROM of the device. This public key pair will not have the same protection scheme as the private key. When the device is deployed, it will now be able to calculate the session key from the start up, based on its own private key and the server's installed public key. Finally, note that it also installs the elliptic curve parameter p, a, b into the device so that the device will be able to just pick random points on the curve to generate different public keys later on, but will not waste resources computing another strong elliptic curve.

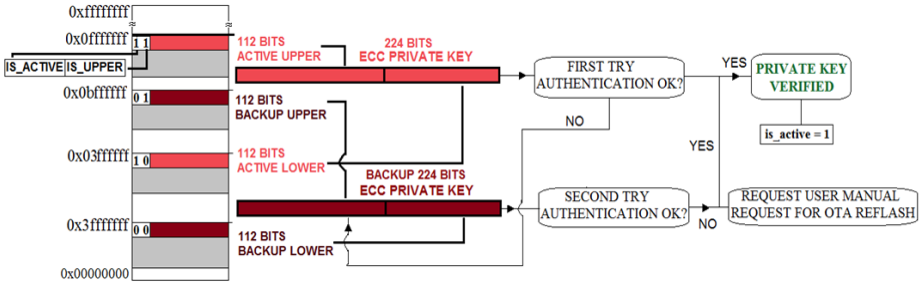

Fig. 7. Memory Hierarchy based Private Key MHPK scheme. In this scheme, we use four 112 bits to protect against malicious attacks in producing a trusted 224 bit key for ECC encryption. After using up the spare key, the device will request user input to re-flash the contents.

Notice that the server only keeps the record of the device's public key but it discards the device's private key, yielding an attack on the server useless against the device itself. Finally, a packet sequence number, which resets after 2^{32} bits for ease of implementation, is used as cryptographic nonce to protect against replay attacks.

Authentication. Authentication protects the devices from man-in-the-middle attacks, as well as providing integrity checks. While typical authentication protocols with public key cryptosystems would require the use of a trusted third party to verify that each other's public key is indeed what is claimed, we choose to not use certificates due to the large overhead of public key infrastructure such as the use of third party certificates. As discussed in section 3, we only concern ourselves with addressing communication between a device and its manufacturer server, and between two devices inside the ecosystem. We choose to authenticate these connections by using a hash-based message authentication code. The HMAC's secret key input is the user's biometrics, whose mechanism will be discussed in detail in the next section. A strong but hardware efficient HMAC like the 128-bit SHA-3 [22] is suggested. Note that a system that authenticates devices belonging to the ecosystem using a secret key generated by the user biometrics can now block man-in-the-middle attacks because a malicious user without access to this key cannot masquerade as one of the devices

belonging to the ecosystem. Likewise, we use this key to digitally sign messages for inter-device communications and between the devices and their OTA provider. Note this supports heterogeneous devices coming from different OTA providers as it is likely to be the case for commercially deployed smart device ecosystems.

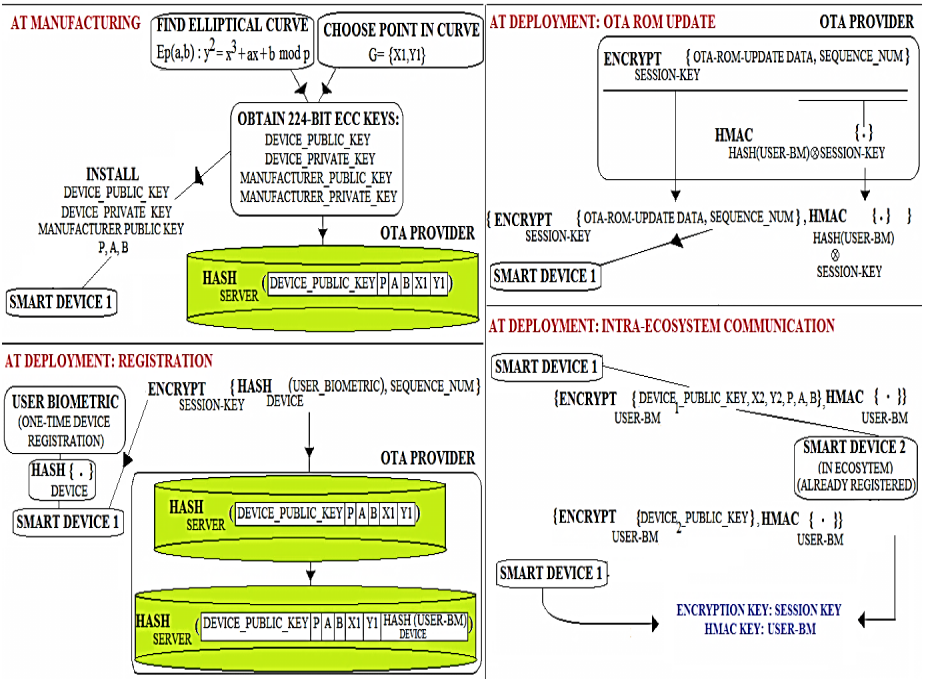

Fig. 8. Device cryptosystem. Notice that it embeds the elliptic curve at installation time. It only encrypts with the ECC session key and provides authentication and integrity guarantees using a HMAC whose key is the hash of the user biometric, common to all ecosystem devices. The HMAC key is the user biometric, common among all the user's devices.

Table 2. Denial of Service protection provided by the framework. Note that when there are only a few failed attempts, the wait time penalty is tolerable, taking into account legitimate failed attempts, but is unforgiving to large numbers of failed attempts.

CONSECUTIVE FAILED MASQUERADING ATTEMPT	WAIT TIME UNTIL NEXT TRY (seconds)
1	2
2	4
3	9
4	16
10	1024 (17 minutes)
100	$1.26e^{10}$ (>>1 year)

Furthermore, note we chose to use an ECC encryption scheme, instead of a symmetric key encryption protocol based on the device private key because the devices will have the different private keys with different OTA providers, and can only communicate with their own OTA server. Similarly, we did not rely on a symmetric key based on the user biometrics because if the biometric reading device were tampered with, it would mean that every device in the ecosystem is hopelessly compromised.

The entire protocol is depicted in figure 8. Finally, to protect against denial of service attacks, unlike the approach in [8] with a linear wait time DoS protection, our algorithm will double the response time after every failed authentication attempt as depicted in table 2, which contains a binary factor for efficient hardware implementation.

4.4 Biometric Authentication

While ASCII-password-based authentication schemes remain the most popular way to let users take control of their systems, economical and practical biometric based systems are starting to be deployed. Biometric based systems are attractive from a user convenience perspective, but they pose unique challenges: (1) If a user's raw biometric data is compromised, there could be cultural and legal repercussions, (2) biometric devices tend to not be 100% accurate, with the possibility of false positives and false negatives, and (3) the user's biometric data must be stored in the device in a protected manner to avoid the compromise of this data.

Instead of the proposed approaches discussed in section 2, ours is a hybrid multi-sensor multi-reading approach that is both ergonomic and commercially viable. Instead of using heterogeneous biometric sensors, we only use fingerprint readings, one of the most economical biometric sensors. In our scheme, the user places all of his five fingers on a palm-sized finger-reading platform, and the system takes at least 3 successful fingerprint readings from each of the fingers. This is because it requires the fingerprint data to be consistent before accepting it, as the rest of the devices will rely on this. Then, after having chosen the fingerprint data, it applies a one-way hash creating a key which will be used for the authentication scheme described in section 4.3.

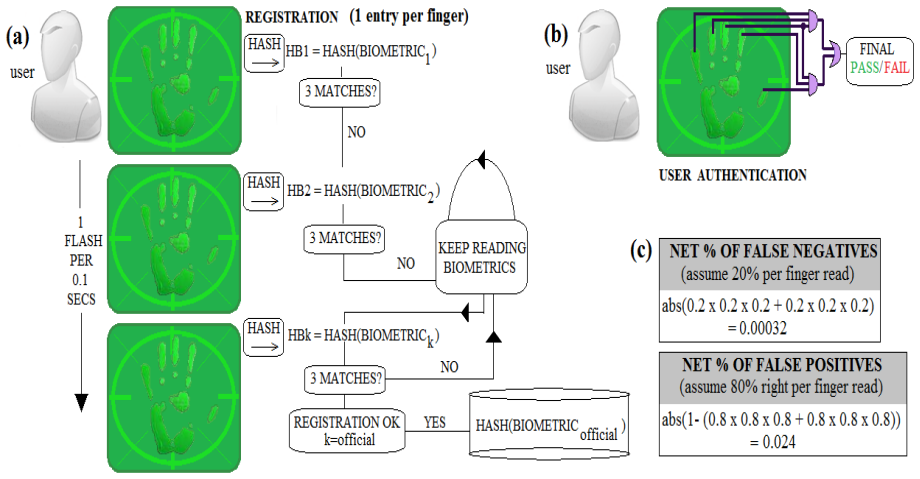

Fig. 9. Registration and user-authentication schemes using economical and ergonomic biometrics. **(a)** For the user registration step, it stores the hash of the user biometric after a reliable result has been found. **(b)** For the authentication step, it performs the AND-OR function. Note its scheme can bump a 20% false negative and positives into a system with negligible failure rates, as seen in part **(c).**

Finally, it allows a user to control the device settings and manual OTA ROM updates by authenticating user manual activity with a scheme that can yield false positives and negatives with near zero probability. The results from first, second and third individual fingerprint reads are AND-ed, while in parallel, the results from the third, fourth and fifth fingers are also AND-ed. Then, these two results are OR-ed together to get the final verdict. Our full biometric based scheme is shown in figure 9. Note that to protect against a malicious user trying to obtain the user biometric data, we never store the data in the device in a raw fashion. Instead, we perform a one-way hash for key generation, and store this same hash to be used for user authentication as well. To verify user authentication, we take the input data, perform the same hash and compare hashes inside the device. Such scheme protects against user privacy attacks on the raw fingerprints themselves.

5 Results

We tested our protocol using a high level simulator that sets up a client acting as the device and two servers, one simulating the manufacturer's OTA provider and the other acting as another device in the ecosystem. Finally, we set up another server acting as a public key infrastructure provider for certain comparison schemes, as will be discussed below. We simulate on four machines, all using Linux centOS 6.4, an Intel Xeon quad core running at a nominal frequency of 2.8 GHz, and 8 GB of RAM. To measure a consistent amount of data sent over the network, we send images made of solely ASCII characters to guarantee a deterministic 1 byte per character ASCII encoding at the network interface. To create more accurate performance comparisons, we maintain the same type of data structures in the code, including encryption and compression algorithms. We measure the performance of our scheme first by measuring the OTA encoding efficiency in terms of the net number of bytes sent and latency for each scheme over a sample software update cycle made of six software updates. We then measure our proposed cryptosystem against an unencrypted system, the de-facto RSA based cryptosystem with a public key infrastructure (PKI), and a similar ECC based cryptosystem with a PKI, using CPU load and latency as the metrics.

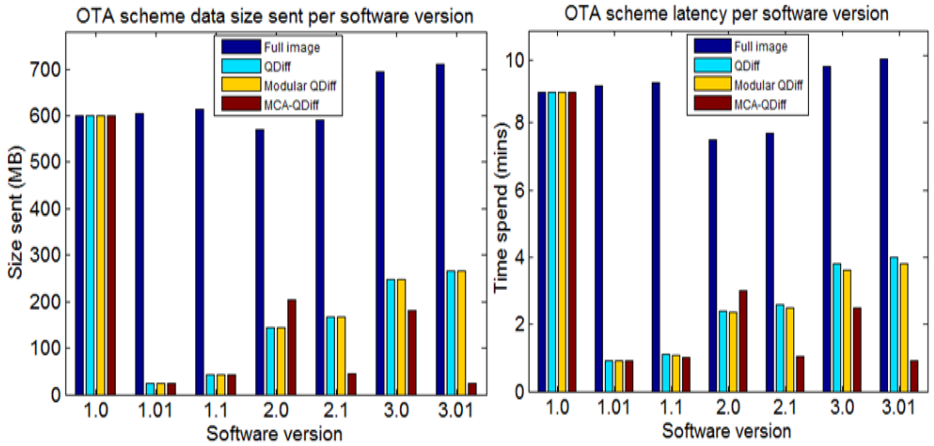

Fig. 10. Comparison of OTA compression schemes: no compression, QDiff, Modular only QDiff, and MCA-QDiff

We first simulate the performance of our OTA protocol against the behavior of the raw QDiff using 600MB images as an input, roughly the expected size of a full OS update [23]. For consistency, all schemes are sent using a biometric-aided ECC encrypted scheme, and all QDiff-based schemes contain a BPE compression step. Figure 10 contains the performance of the OTA protocols given the versioning scheme pattern, comparing four possible scenarios: (1) sending the full image per release, (2) using QDiff, (3) using the *modular* memory mapping enabled QDiff algorithm with a reduced search space, and (4) using our full MCA-QDiff scheme. Note that the sent bytes do not scale perfectly with transmission delay per byte because QDiff based schemes spend time compressing and decompressing with BPE. Furthermore, notice that modular QDiff performs just as QDiff for small deltas, but for larger code deltas, Modular QDiff outperforms the raw QDiff. Finally, as new software versions come out, the code deltas found by QDiff steadily increase, due to lack of self-cleaning and accumulated slop spaces. It is only after the device has gone through enough changes that the advantages of MCA-QDiff become significant. MCA-QDiff obtains an average latency performance improvement of 15% over a sample update cycle.

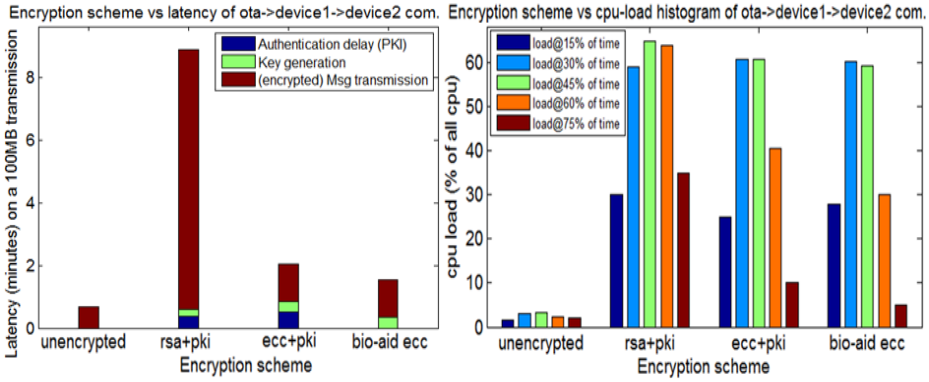

Fig. 11. Comparison of cryptosystem schemes: Unencrypted system, a standard 2048 bit RSA-encrypted system with a PKI, a 224 bit ECC-encrypted system with a PKI, and a 224 bit ECC encrypted system using the biometric for signature instead of a PKI.

To compare the efficiency of our cryptosystem, we implemented four different schemes: (1) sending the unencrypted image; (2) sending the encrypted image using the RSA protocol with a PKI; (3) sending the encrypted image using ECC with a PKI; (4) and finally, sending the encrypted image with our ECC method that bypasses the need of a PKI. We choose to send a 100MB image, the typical size of an OTA iOS update [23]. To emulate a public key infrastructure, we require that each communicating party contact the fourth sever, establish an independent encrypted channel, and have that server search a very large database for a matching key. Notice that when implementing an algorithm, we do so efficiently by exploiting several modulus operation rules, Euclid's algorithm and Fermat's test for compositeness [24], among others. Furthermore, when choosing an ECC curve in our implementation, we pick an arbitrary elliptic curve. However, note that the NST has published a set of "safe" curves [25], and hence on a real implementation, they would be preferred. Figure 11 shows

the full comparison in terms of absolute latency and through a CPU utilization histogram captured at five points during the execution of the simulation. Note that while the unencrypted image has the lowest load and latency on the system, its lack of security is intolerable. Our scheme shows the best balance between performance and security. In particular, our scheme shows a 25% latency reduction over a vanilla ECC system, and a 5% lower average load. Coupled with a 15% latency reduction in our OTA protocol, we obtain a net 35% reduction in latency. Finally notice that our biometric-aided scheme is not only faster, but will also have several other benefits not accounted for in this test bench because a typical public key infrastructure requires additional hardware use and protection schemes, since it is an added source of vulnerability.

6 Conclusion

This paper presented an integrated security framework for a smart device ecosystem. This ecosystem poses unique challenges because the devices are interconnected in a similar manner to wireless sensor networks yet their OTA-ROM provider model is similar to that of smart phones. Furthermore, they have unique characteristics such a level of user interaction much less prominent than smart phones, and yet much more so than conventional WSNs. Updating their software and firmware is done over the air, adding another layer of vulnerability but also opening several optimization areas. Thus, we have presented a system to address this imminent security challenge through design modularity, hardware and software co-design, and finally, through computation and cost efficient design choices. As these smart devices become widely deployed, they will start forming ad-hoc networks on household and industrial settings, meanwhile potential security breach points as well as their treat level will increase exponentially. Hence, designers looking to successfully deploy these ecosystems should turn towards a strong, yet efficient, security framework.

References

1. Vermesan, O., Friess, P., Guillemin, P.: The Internet of Things - Strategic Research Roadmap. In: Cluster of European Research Projects on the Internet of Things, CERP-IoT (2009)
2. Linux Foundation: Tizen OS (2012), `https://www.tizen.org/`
3. Oommen, P.: A Framework for Integrated Management of Mobile-Stations Over-the-Air. In: IEEE/IFIP International Symposium on Integrated Network Management Proceedings (2001)
4. Cong Vo, C.: A Framework for Over the Air Provider-initiated Software Deployment on Mobile Devices. In: 19th Australian Conference on Software Engineering, ASWEC (2008)
5. Ling, Y., Tiansheng, H., Caixing, L., Yue, X., Haoen, Z.: A reprogramming protocol based on state machine for wireless sensor network. In: International Conference on Electrical and Control Engineering, ICECE (2010)
6. Brown, S., Sreenan, C.J.: A New Model for Updating Software in Wireless Sensor Networks. IEEE Network, 42–47 (2006)
7. Bing, B.: A Fast and Secure Framework for Over-the-Air Wireless Software Download Using Reconfigurable Mobile Devices. IEEE Communications Magazine, 58–63 (2006)

8. Bauer, J., Bieling, J., Bothe, A., Schwamborn, M.: Selective and Secure Over-The-Air Programming for Wireless Sensor Networks. In: 21st International Conference on Computer Communications and Networks, ICCCN (2012)
9. Nilsson, D., Larson, U.E.: Secure Firmware Updates over the Air in Intelligent Vehicles. In: IEEE International Conference on Communications Workshops, ICC Workshops (2008)
10. Chiang, M., Lu, T.: Two-Stage Diff: An Efficient Dynamic Software Update Mechanism for Wireless Sensor Networks. In: IFIP 9th International Conference on Embedded and Ubiquitous Computing, EUC (2011)
11. Bin Shafi, N., Ali, K., Hassanein, H.S.: No-reboot and Zero-Flash Over-the-air Programming for Wireless Sensor Networks. In: 9th Annual IEEE Communications Society Conference on Sensor Mesh and Ad Hoc Communications and Networks, SECON (2012)
12. Shibata, Y., Kida, T., Fukamachi, S.: Byte Pair Encoding: a text compression scheme that accelerates pattern matching. Technical report DOI-TR-161, Kyshu University (1999)
13. Kiyohara, R.: A New Method of Fast Compression of Program Code for OTA Updates in Consumer Devices. IEEE Transactions on Consumer Electronics, 812–817 (2009)
14. Barker, E., Barker, W., Burr, W.: Recommendation for Key Management. Part 1: General, NIST Special Publication 800-57 (2007)
15. Gupta, K., Silakari, S.: ECC over RSA for Asymmetric Encryption: A Review. IJCSI International Journal of Computer Science Issues, 370–375 (2011)
16. Ganesan, S.: An Efficient Protocol for Resource Constrained Platforms Using ECC. International Journal on Computer Science and Engineering, 89–91 (2009)
17. Chen, D., Nixon, M., Lin, T.: Over the Air Provisioning of Industrial Wireless Devices Using Elliptic Curve Cryptography. In: IEEE International Conference on Computer Science and Automation Engineering, CSAE (2011)
18. Gnanasivam, P.: Ear and Fingerprint Biometrics for Personal Identification. In: International Conference on Signal Processing, Communication, Computing and Networking Technologies, ICSCCN 2011 (2011)
19. Huang, Y., Ao, X., Li, Y.: Multiple Biometrics System based on DavinCi Platform. In: International Symposium on Information Science and Engineering, ISISE (2008)
20. Zhang, Y., Sun, D., Qiu, Z.: Hand-Based Feature Level Fusion for Single Sample Biometrics Recognition. In: International Workshop on Emerging Techniques and Challenges for Hand-Based Biometrics, ETCHB (2010)
21. Nilsson, D., Sun, L., Nakajima, T.: A Framework for Self-Verification of Firmware Updates over the Air in Vehicle ECUs. In: IEEE GLOBECOM Workshops (2008)
22. Guo, X., Huang, S., Nazhandali, L.: Fair and Comprehensive Performance Evaluation of 14 Second Round SHA-3 ASIC Implementations. In: NIST 2nd SHA-3 Candidate Conference (2010)
23. iOS OTA versioning data, XML format. Apple Inc. (2013), `http://mesu.apple.com/assets/com_apple_MobileAsset_SoftwareUpdate/com_apple_MobileAsset_SoftwareUpdate.xml`
24. Euler, L.: Theorematum quorundam ad numeros primos spectantium demonstratio. Commentarii Academiae Scientiarum Petropolitanae 8, 141–146 (1741)
25. National Institute of Standards and Technology: Recommended elliptic curves for federal government use (1999), `http://csrc.nist.gov/CryptoToolkit/dss/ecdsa/NISTReCur.pdf`