A Secure Supply-Chain RFID System that Respects Your Privacy

A prototype RFID system uses public-key cryptography to simplify deployment, reduce trust issues between the supply-chain owners and tag manufacturers, and protect user privacy. It demonstrates that high-security and standard EPC tags can coexist and share the same infrastructure.

Alex Arbit
Tel-Aviv University

Yossef Oren
Columbia University

Avishai Wool
Tel-Aviv University

Introducing RFID tags based on the Electronic Product Code (EPC) standard[1] into the global supply chain is one of the world's most ambitious pervasive computing projects. EPC tags are low-cost wireless devices that associate a computer-readable ID to physical objects. They were designed as an upgrade to the familiar 14-digit Universal Product Code (UPC) bar code. In contrast to UPC bar codes, which signify only that an item is of a particular type, EPC tags can carry more data, including a 96-bit globally unique item identifier (UII) for each individual tagged item. EPC tags communicate with tag readers using UHF radio, letting readers interrogate them from a distance of several meters—even with no line of sight between the tag and the reader.

Although the increased data capacity and interrogation range make EPC tags useful for supply-chain operators, these same capabilities pose privacy issues that didn't exist with optical bar codes.[2] For example, by placing EPC readers in multiple locations and searching for any RFID tags a person carries, such as RFID-tagged clothes or banknotes, it becomes possible to track individuals moving between the locations. In a broad survey of the RFID security and privacy threat landscape, Pawel Rotter argues that the demand for security increases with the potential damage to users and the potential gain to attackers.[3] Encrypting the communication between tags and readers would allow only a legitimate reader to read a tag's UII and thus prevent eavesdroppers from learning anything from a recorded session.

Low-cost RFID tags are generally susceptible to reverse engineering, and a captured tag can also be cloned and attached to many counterfeit items. A *track-and-trace* network is a basic anticounterfeiting technique that links all of a supply chain's EPC readers to a central server. Every time a reader that's connected to the network accesses a tagged item, the item's current physical location is tracked and updated on the central server. A counterfeited item will exhibit an unusual and detectable usage pattern, such as being present in two different countries at the same time, and can be blacklisted. To avoid blacklisting, a more sophisticated attacker will create fake tags that return different arbitrary or randomized UIIs. This requires a further anticounterfeiting mechanism: cryptographically signed UIIs that prevent attackers from generating their own arbitrary UIIs.

Published by the IEEE CS ■ 1536-1268/14/$31.00 © 2014 IEEE

Because of the very low resources available on EPC tags, public-key cryptography is generally considered too complicated to deploy on tag hardware—leaving symmetric-key cryptography as the primary option for such systems. Unfortunately, symmetric-key cryptography imposes negative trust issues between the various stakeholders. Furthermore, if a single tag is reverse-engineered and the symmetric key is leaked, the whole system's security collapses. Here, we present an approach to secure the EPC supply chain using public-key cryptography to protect the interests of all stakeholders: supply-chain owners, merchants, and consumers.

System Design

The crucial element in our system is the WIPR (Weizmann-IAIK [Institute for Applied Information Processing and Communications] Public Key for RFID) encryption scheme (see the sidebar).[4] WIPR has a low resource footprint—full-strength WIPR encryption can be implemented even within the limited resources of an EPC tag. We created a full system to demonstrate our approach, including software usable by various entities in the supply chain, a fully EPC-compliant UHF tag implementing WIPR in custom firmware, and a prototype point-of-sale (POS) terminal that uses an off-the-shelf RFID reader and supports both encrypted and unencrypted tag communication while strictly operating within the standard EPC air-interface specification.

Figure 1 shows a logistic view of our system, representing the members of a secure RFID supply chain: a supply-chain owner who wishes to use RFID technology but doesn't manufacture tags; the tag manufacturer that produces and deploys many tags on behalf of the supply-chain owner; the merchant, who owns the POS terminal connected to an individual reader *j*; and an individual RFID tag *i*. Each tag is attached to a particular piece of merchandise in the system. The reader is the device that communicates with the tags.

The parties' interests differ. The supply-chain owner requires UII signatures that can't be forged and UIIs that no other party, including the manufacturer, can sign. The owner also requires that if and when tags are reverse-engineered and all their private data is extracted, the attacker can't do more than clone the specific tags that were reversed. In particular, a tag's private data should not allow the attacker to generate new signatures on UIIs.

The manufacturer needs access to the signed UIIs as well as the cryptographic keys that must be written to the tags. The merchant's reader must be able to communicate with the tags and verify the UII signatures. Finally, the customer holding the tag needs assurance that adversaries can't eavesdrop on the tag's communication with the reader and so requires the communication between tag and reader to be encrypted. Further, the customer needs assurance that a rogue reader won't be able to either read the tag's data or identify and track the tag.

To meet these conflicting requirements, our system uses two pairs of public-private keys:

- a private-signing key, k_S, together with its public-verification key, k_V, and
- a private-decryption key, k_D together with its public-encryption key, k_E.

It uses the pair (k_S, k_V) to sign and verify UIIs and the pair (k_D, k_E) to authenticate the reader to the tag and to encrypt and decrypt the communication between them. The supply-chain owner generates all the key material. The signing key k_S never leaves the supply-chain owner's premises. Instead, the supply-chain owner generates a list of signed UIIs and sends them to the tag manufacturer, together with the public-encryption key k_E. Without the private-signing key, the tag manufacturer can't create arbitrary signed UIIs—in other words, the owner doesn't need to trust the manufacturer. The manufacturer produces all the tags and embeds within each tag the public-encryption key k_E, and a single signed UII from the list.

The use of public-key cryptography guarantees that an adversary gains nothing from reverse engineering an individual tag other than the single signature for that individual tag's UII and a public key that lets it send encrypted messages to the reader but not decrypt them. Finally, the supply-chain owner sends the decryption key k_D and the signature-verification key k_V to the merchant's reader, so the reader can decrypt the tag messages and verify the UII signatures.

Data Flow and Communication Protocol

Figure 2 shows our system's data flow and communication protocol in the actions of three parties: the supply-chain owner's offline key generation and signed UII generation and the communication protocol between the reader and an individual tag. The manufacturer doesn't appear in the figure because it doesn't participate in the communication protocol.

The reader starts the protocol by activating the tag and sending it a random challenge. The tag responds by encrypting this challenge together with the tag's signed UII and additional random bytes, then sending this data back to the reader. The tag's response also authenticates the reader to the tag because a rogue reader, lacking the decryption key k_D, can't decrypt it. Finally, the reader decrypts the tag's response, verifies that it contains the correct challenge and UII signature, and outputs the UII.

This scheme has several beneficial properties.

First, the use of encryption means that the adversary can't discover the UII or the UII's signature by intercepting over-the-air communications. Unless the adversary knows how to crack the

What is WIPR?

Yossef Oren and Martin Feldhofer developed the WIPR (Weizmann-IAIK [Institute for Applied Information Processing and Communications] Public Key for RFID) low-resource, public-key encryption scheme.[1,2] Jiang Wu and Douglas Stinson subsequently proposed an improvement that claims to reduce hardware requirements and protect against some attacks.[3]

WIPR is a variant of Rabin's encryption scheme,[4] which is provably as secure as factoring large numbers. In Rabin's scheme, the private key consists of two large primes p and q, which are multiplied to form the public key $n = p \cdot q$. To encrypt a message m, the sender calculates its square and reduces it modulo n: $c = m^2$ (mod n).

To decrypt a ciphertext, the receiver calculates the square roots of c modulo p and q, then combines the resulting values using the Chinese Remainder Theorem. Each ciphertext has two possible roots modulo p and two roots modulo q ($\pm m$ (mod p) and $\pm m$ (mod q), leading to four possible plain texts for each ciphertext. To allow the receiver to determine which plaintext is the correct one, the sender typically adds some redundancy to the message (in the work reported in the main article, the reader challenge serves this purpose).

The encryption element of Rabin's scheme is relatively easy to implement, requiring only a single multiplication and modular reduction. However, modular reduction is a RAM-intensive process, a fact that limits the applicability of Rabin's algorithm to memory-constrained devices. To solve this, Adi Shamir[5] (and simultaneously David Naccache[6]) suggested replacing the modular-reduction step by an addition of a large random multiple of n, where $|r| > |n| + 80$: $c = m^2 + r \times n$.

The decryption algorithm is identical to Rabin's original scheme. Shamir proved the equivalence of this scheme's security and the original Rabin scheme. The reduced scheme is easier to implement because it has only multiply and accumulate operations and no modular reductions. The multiply-accumulate algorithm has a very low RAM footprint when compared to standard modular reduction, because it doesn't need to store the entire ciphertext in memory. Instead, a multiplication-by-convolution algorithm can be used to calculate small chunks of the output one after the other in an accumulator register, then transmit them as soon as they are ready.

One drawback of this method is the increase in ciphertext size. In the original Rabin algorithm, the ciphertext size was n bits, but this resource-reduced implementation increases the output size to $n + r \geq 2n + 80$ bits.

Shamir's version of Rabin's scheme replaces the challenge of storing with the challenge of storing the large random number r. However, because r is written-to only once per protocol execution, it can be stored in EEPROM storage, which is plentiful on smart cards.

EEPROM is generally not available on low-cost RFID tags, so Oren and Feldhofer developed a way to remove this additional storage requirement.[1] The WIPR scheme replaces the random value r with the output of a low-resource reversible stream cipher.

WIPR is a probabilistic scheme that uses randomness to increase its security. The reader issues a different random challenge each time it enters the WIPR protocol. This makes it very difficult for an adversary to counterfeit a tag by recording its response and replaying it later. The tag also uses a different random string in each response, making it difficult for an adversary to track a tag by sending the tag a similar challenge multiple times and looking for a repeated response.

The plaintext payload m consists of the 80-bit random reader challenge, the 96-bit EPC unique item identifier (UII), the 320-bit ECDSA signature for the UII, and 528 additional random bits. Because the RSA and Rabin cryptosystems both rely on the same fundamental complexity assumption, a security level comparable to 1,024-bit RSA also requires a 1,024-bit Rabin key, resulting in a ciphertext size of $2 \times 1{,}024 + 80 = 2{,}028$ bits.

REFERENCES

1. Y. Oren and M. Feldhofer, "WIPR—Public-Key Identification on Two Grains of Sand," *Proc. Workshop on RFID Security 2008, 2008,* pp. 15–27.
2. Y. Oren and M. Feldhofer, "A Low-Resource Public-Key Identification Scheme for RFID Tags and Sensor Nodes," *Proc. 2nd Int'l Conf. Wireless Network Security* (WiSec 09), ACM, 2009, pp. 59–68.
3. J. Wu and D.R. Stinson, "How to Improve Security and Reduce Hardware Demands of the WIPR RFID Protocol," *Proc. 2009 IEEE Int'l Conf. RFID* (RFID 09), 2009, pp. 192–199.
4. M. Rabin, *Digitalized Signatures and Public-Key Functions as Intractable as Factorization,* tech. report, MIT, 1979.
5. A. Shamir, "Memory Efficient Variants of Public-Key Schemes for Smart Card Applications," *Proc. Advances in Cryptology* (EuroCrypt 94), LNCS 950, 1995, p. 445.
6. D. Naccache, Method, *Sender Apparatus and Receiver Apparatus for Modulo Operation,* European Patent Application 91402958.2, filed 27 Oct. 1992.

encryption scheme, the only practical way to recover the tag's payload would be to reverse-engineer the tag or compromise a reader.

Second, the reader issues a fresh random challenge in every protocol execution. This foils an adversary that tries to impersonate a tag by recording challenge-response pairs used in successful transactions and waiting for a reader challenge to be repeated. The adversary would have to record an impractically

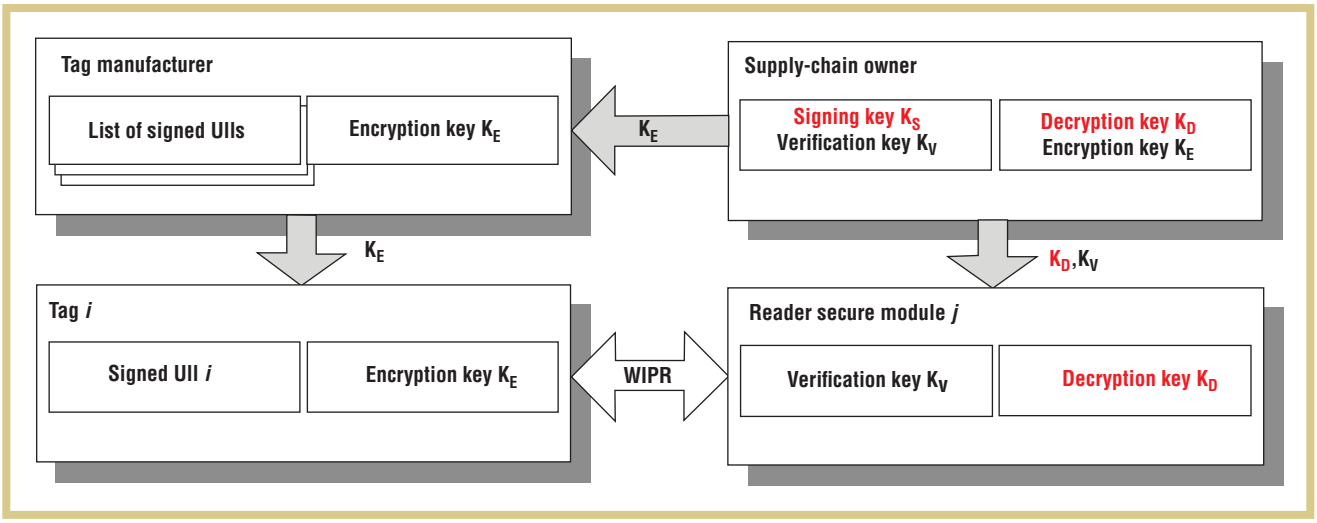


Figure 1. A logistic view of the proposed public-key-based supply-chain system based on unique item identifiers (UII) and the WIPR encryption scheme. Its members include a supply-chain owner, a tag manufacturer, a secure reader module representing a merchant's point-of-sale terminal, and an individual RFID tag. Private keys (signing and decryption) are shown in red.

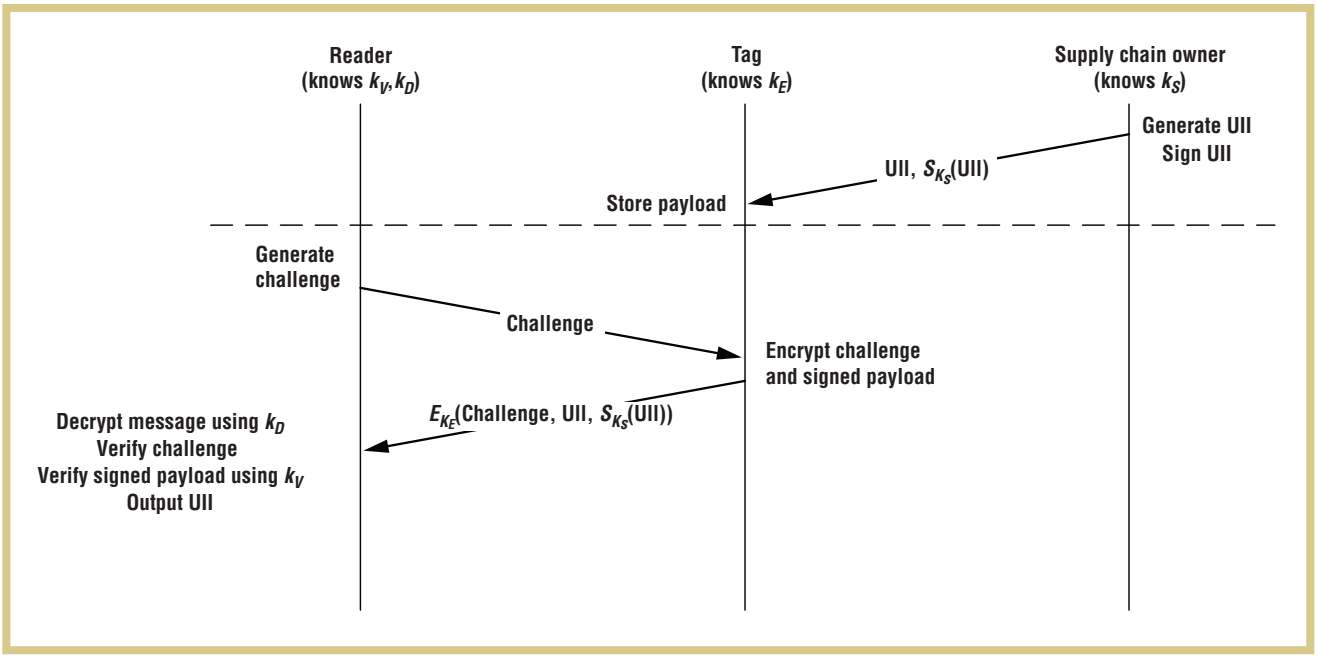


Figure 2. A secure RFID supply-chain system based on public-key cryptography. The merchant's reader initiates the communication protocol by activating the tag and sending it a random challenge. (Note that "E()" stands for the encryption function.)

large number of transactions before the same challenge is repeated.

Third, the tag adds different random bytes to its response every time it's queried. This foils an adversary trying to track a certain tag (that it knows from a previously recorded challenge-response pair) by masquerading as a reader, repeatedly sending the same challenge to the tag, and checking whether the tag returns the previously recorded response packet. The adversary would have to engage a tag in an impractically large number of transactions with the same challenge before the tag generates the same response twice.

Fourth, even though a reverse-engineered tag can be cloned, the only tag information of interest in cloning is a single signature, $S_k(UII)$, which can only be used with a specific UII. (Note that Sk_S(UII) means the message "UII" is signed using the signing key k_S.) Thus, even an adversary who

has physically probed the tag to discover its full payload value can't forge a new tag with a different UII. This effectively creates a *break-once-run-once* situation for tags, such that counterfeited merchandise must bear a limited number of well-known IDs that can be more easily tracked and blacklisted.

Finally, a compromised reader can't be used to forge new tags because it has only the verification key and not the signing key. A compromised reader can, however, be used to track tags from any vendors for which it has the keys.

Implementation Choices

To implement our system, we first selected public-key signature and encryption schemes that offer suitable security while staying within the stringent hardware requirements of low-cost EPC tags. Because EPC tags are passively powered by the reader, we had to minimize their energy consumption to maximize their usable range. In practical terms, this restricted the total gate budget of RFID tag chips, including security functionality, to approximately 10,000 NAND gate equivalents (GEs).[5] Strong cryptographic elements, especially of the public-key variety, have been traditionally considered too complicated for this limited gate budget.[6] The limited gate budget also imposes a severe restraint on the amount of RAM and ROM we can place on the tag.

Because the UII is actually signed outside the tag and only stored in the tag's memory, we weren't concerned with the signature scheme's implementation cost but only with its storage cost. So we searched for a public-key signature scheme with relatively short signatures and finally chose the Elliptic Curve Data Security Algorithm (ECDSA). For a security level comparable to 1,024-bit RSA, the ECDSA cipher uses a 160-bit key and generates 320-bit signatures.

We chose WIPR as the public-key encryption scheme for communication between the reader and the tag because it has one of the smallest chip-area requirements of any public-key encryption scheme, making it simple enough to place on a low-cost RFID tag.[4] It also has a relatively large payload size (almost 1,000 bits per ciphertext), making it versatile enough to support our specific application. For a security level comparable to 1,024-bit RSA, the WIPR protocol generates a 2,208-bit ciphertext.

Our next challenge was to introduce the cryptographic challenge-response protocol into the preexisting EPC air interface.[1] Our protocol requires sending a challenge to the tag and verifying its response. Because the EPC specification doesn't specifically include challenge-and-response messages, we needed to find a way to retrofit the existing infrastructure so that it would support our system. Furthermore, WIPR messages are relatively long and must be split into parts to be transmitted and then reassembled by the reader.

As suggested by Daniel Bailey and Ari Juels,[7] we implemented the challenge-response protocol using *memory-mapped I/O* and regular EPC messages. Writing the challenge to a certain region in the tag's memory activates the WIPR protocol, while reading the response from another region invokes the actual encryption operation. As described in the sidebar, the WIPR ciphertext can be calculated byte by byte, reducing the amount of physical memory required to store the encryption result.

Figure 3 illustrates our protocol implementation. The left side shows how a standard (non-WIPR) tag identifies itself to a non-WIPR RFID reader. In contrast to normal tags, the WIPR tag does not transmit its UII over the air unencrypted, which would allow the tag to be tracked by rogue readers. Instead, it returns a special semi-random ID that partially identifies the tag while indicating that this tag supports the WIPR protocol. This ID is created by splitting the 96-bit EPC address space into both a fixed part and a random part that is recalculated every time the RFID tag reboots. If the highest degree of privacy is required, the fixed ID part could provide no information about the tagged item, other than the fact that it supports WIPR. It's also possible to sacrifice some privacy for utility by embedding a limited amount of information about the tagged item in the fixed part. For example, the fixed ID part could contain the tagged item's type without explicitly identifying it, just like a standard UPC optical barcode. Henry Holtzman and his colleagues describe a similar privacy-protection method using pseudo-IDs.[8]

If the reader doesn't support WIPR, the inventory process concludes at this stage with the reader having only partial knowledge of the item's identity, thus preventing the user from being tracked. If the reader supports WIPR, it will proceed by writing the challenge to a special memory area on the tag, then reading the encrypted response from another memory area. The reader will then be able to decrypt the response using its stored WIPR private key, thus obtaining the full UII and precisely identifying the tagged item.

In Figure 3, $T_{challenge}$ is the time it takes the reader to send the challenge to the tag. $T_{encrypt}$ is the time it takes the tag to encrypt, and $T_{response}$ is the time it takes the tag to send its response back to the reader. $T_{challenge}$ and $T_{response}$ are determined by the link speed between the tag and the reader, but $T_{encrypt}$ is solely a function of the WIPR algorithm's implementation. Only a part of $T_{response}$ (marked $T_{response'}$) happens after encryption is completed. As we discuss in the sidebar, this is due to a special property of the WIPR algorithm that allows the ciphertext to be generated byte by byte.

We evaluated RFID readers from several vendors and discovered that they all shared a common design, consisting of a firmware module and associated host software. The firmware module handles the EPC air interface and state machine, and communicates with the software running on the host computer using a relatively high-level protocol. Although modifying the host computer's software is relatively simple, updating the reader's

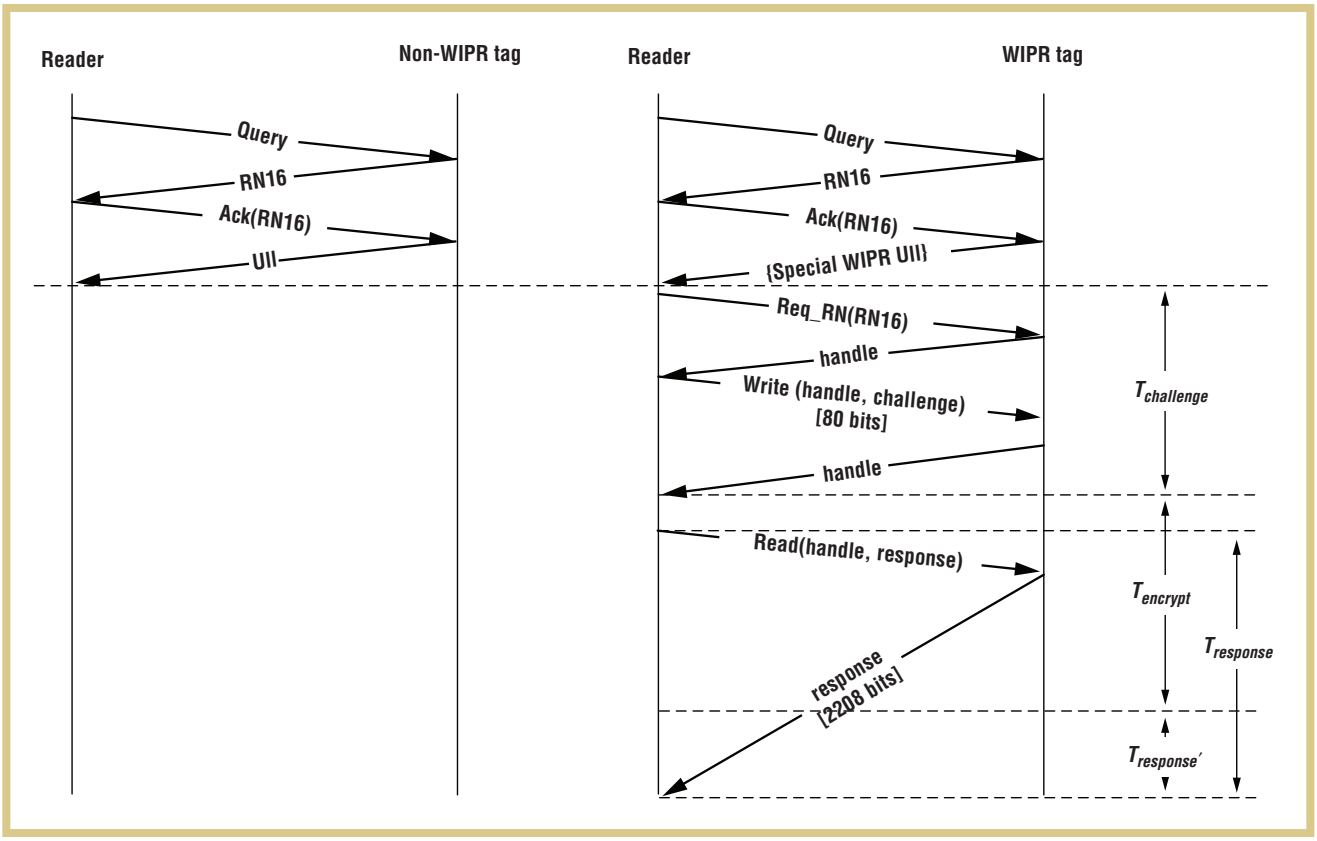


Figure 3. Implementing cryptographic challenge-response using standard EPC commands. The WIPR tag does not transmit its UII over the air unencrypted, but instead returns a special semirandom ID that partially identifies the tag while indicating that this tag supports the WIPR protocol. $T_{challenge}$ is the time it takes the reader to send the challenge to the tag; $T_{encrypt}$, the time it takes the tag to encrypt; and $T_{response}$ the time it takes the tag to send its response back to the reader. (Note that RN16 is a 16-bit random number.)

embedded firmware module is more complex and sometimes impossible. By using only standard memory-write and memory-read commands, which are mandatory in the EPC specification and as such already supported by the firmware, we made sure that standard off-the-shelf reader firmware will immediately support our new protocol. To support WIPR, stores only need to update the software running on the POS terminal.

Prototype Implementation Details

After concluding the design, we implemented the scheme and measured its performance using real EPC equipment.

Figure 4 shows the system setup. It consists of several standard off-the-shelf EPC-compliant RFID tags, a standard off-the-shelf RFID reader (the CAEN RFID reader), and several PC software elements that support WIPR.

A fully deployed solution would use an inexpensive ASIC-based passive tag, but we were interested in a system optimized for adaptability, allowing easy and fast prototype development. For this purpose, we selected the IAIK UHF Demotag, a hardware-prototyping platform.[9] While the Demotag is battery-powered, it behaves like a passive EPC-compliant tag. It features an ATMega128 microcontroller with a programming interface that uses the Joint Test Action Group (JTAG) standard and in-system programming. In addition to the radio frequency (RF) interface, the Demotag also supports a serial interface, which we used to configure the tag. We developed the tag software on a Linux workstation, using Rowley CrossStudio for AVR.

We selected the CAEN DK828EU reader, because it's relatively easy to control in Matlab and conforms with European Telecommunications Standards Institute's power requirements. To calculate the reader's average read rate, we measured how long it took the reader software to read different-sized buffers from the tag's general-purpose memory bank using EPC C1G2 BLOCK_READ commands. By measuring the difference in response times for differently sized blocks (varying between 8 and 272 bits), we were able to measure the read rate while keeping constant any additional

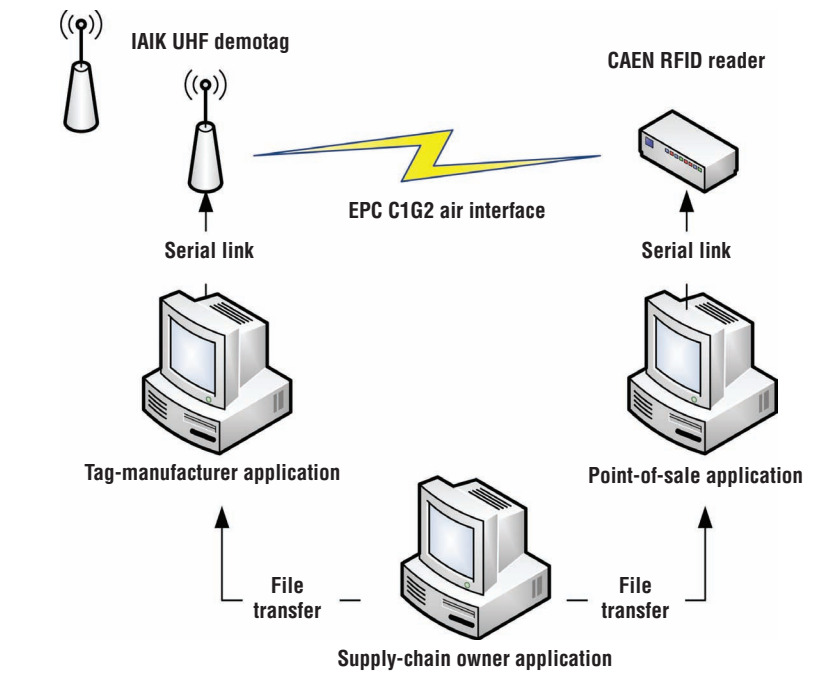


Figure 4. System implementation setup. It consists of standard off-the-shelf EPC-compliant RFID tags, a standard off-the-shelf RFID reader, and WIPR-compatible PC software elements to support the supply-chain owner, tag-manufacturer, and POS applications.

delays related to the operation, such as radio power-on/power-off times, propagation delays, and the execution of the EPC C1G2 singulation protocol. Our tests showed that our reader had an average read rate of approximately 15 kilobits per second, a fact that dominated our system's overall performance.

On the PC side, we wrote a software suite to deliver functionality to each of the supply chain's three members as identified in Figure 1:

- The supply-chain-owner management application creates encryption and signature key pairs and signs UIIs with the private-signing key.
- The tag-manufacturer management application imports lists of signed UIIs and burns them into blank tags.
- The POS application is simple to use and implements the WIPR protocol at the point of sale.

The information exchanged among the three programs is limited, exemplifying the limited trust the system requires. The tag-manufacturer application receives only the public-encryption key k_E and a list of signed IDs, while the POS application receives the private decryption key k_D and the public-verification key k_V.

To use our system, the supply-chain owner first generates the encryption and signature key pairs, then imports a database of items with unsigned UIIs (provided in the real world from a logistics software) and signs them using the private-signing key k_S. Finally, the supply-chain owner creates two files: one containing k_E and a list of signed UIIs (to be sent to the tag manufacturer), and the other containing k_D and k_V (to be sent to the POS terminal).

The tag manufacturer loads the file from the supply-chain owner, then connects a sequence of blank tags to the tag manufacturer's workstation. Each tag is programmed with the public-encryption key k_E and a single signed UII. Other than by performing the WIPR protocol, there is no command that extracts this signed payload from the tag unless it is reverse engineered.

We designed the POS application to be as similar as possible to conventional POS terminal software. It features a simple GUI that's usable on a touchscreen. The software continuously scans for RFID tags in its vicinity. When it finds a tag, it displays the tagged item's picture and lists its price. If the tag is identified as a WIPR tag, the software carries out the WIPR protocol and optionally sounds an alarm if the authentication process fails.

A video demonstration of our POS system, including key creation and distribution and POS operation, is available online at http://youtu.be/ZFrT1xRTorE. As the video shows, the system works well in a mixed-tag environment. From the user's viewpoint, WIPR and non-WIPR tags exhibit the same behavior, other than a slight delay due to the WIPR protocol processing.

Performance Evaluation

In earlier work,[4] we showed that an ASIC implementation of WIPR has an acceptable gate count (approximately 4,700 GEs) and power consumption (mean current draw of 14 μA). For comparison, Martin Feldhofer and his colleagues described an Advanced Encryption Standard symmetric key cipher implementation with a gate count of 3,400 GEs and a mean current draw of 3 μA.[10]

To learn whether the cryptographic operation is indeed an inherent bottleneck or whether it can be sped up enough to make the system usable, we considered the Demotag's general-purpose 8-bit microcontroller to be inherently slower than a custom ASIC implementation. Indeed, a naive software WIPR protocol implementation that was functionally identical to the ASIC's implementation took an unacceptable

7 seconds to perform encryption. However, as we've shown elsewhere in more detail,[11] we were able to speed up the software implementation by two orders of magnitude.

We accelerated WIPR by caching two encryption components: the random padding r (1,104 bits) and the payload m (496 bits). We evaluated three possible scenarios:

- the naive implementation, which doesn't cache the random padding of m and the long random number r, but instead recalculates them on demand;
- caching m in SRAM; and
- caching the values of both m and r.

We discovered that caching data in SRAM has a dramatic effect on the execution time. The first scenario required 7 seconds to encrypt; the second scenario took 1.18 seconds; and the third scenario took just 180 ms.

Other than encryption itself, we found another serious bottleneck in communication, with the dominant parameter being the number of roundtrips made by the reader. The particular reader we used didn't recognize the concept of sessions and repeated the RFID singulation process with the tag every time it sent a command. This behavior significantly slowed down the protocol, so sending the 80-bit challenge took 200 ms and reading back the encrypted response took 460 ms. However, we can reduce both of these times significantly.

First, we could improve the time immediately with better use of the air interface. By sending the challenge in a single 80-bit packet and keeping the tag in the SECURED state, $T_{challenge}$ can decrease from 200 ms to an estimated 85 ms. Next, keeping the tag powered in the SECURED state throughout the response phase would remove the unnecessary singulation steps and save even more time. Finally, it's possible to pipeline the encryption and response transmission. Using WIPR, the tag can compute the ciphertext in small blocks and send them to the reader as soon as they're ready. The total time to perform the entire protocol in this case is equivalent to the time required to power on the tag and send it a challenge (85 ms), the time required for the tag to calculate the full response (180 ms), and the time required to send the final chunk, which is ready only after encryption is finished (60 ms). Under these minor modifications, we estimate the entire protocol (including both identification and authentication) will take 325 ms.

For a more dramatic optimization, the entire 276-byte response can be read by a single read command, to be issued immediately after the challenge is sent. This is possible because the tag can be designed to concurrently backscatter the ciphertext's initial bytes while it calculates the following ones. Because the data link takes only 112 ms to transfer 2,208 bits, the entire protocol time is dominated in this case by $T_{encrypt}$, leading to a total estimated time of 265 ms for the entire protocol. For comparison, the execution time of a standard query-response is approximately 40 ms.

WIPR-enabled tags show a practical design for a secure RFID supply-chain system that uses public-key cryptography. They are fully compatible with the existing ecosystem of nonsecure tags, readers, and terminals. Their use of public-key cryptography reduces trust issues between the supply-chain owner and tag manufacturer, ensures that reverse-engineered tags do not compromise the whole system's security, and protects user privacy. We conclude that the public-key approach is a viable design alternative for supply-chain RFID EPC tags. P

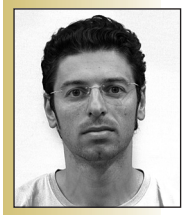
ACKNOWLEDGMENTS

Yossef Oren was the corresponding author for this article. Shay Cohen, Amit Erez, and Doron Shutzberg wrote the point-of-sale software implementation.

REFERENCES

1. M. Aigner, T. Plos, and A.R.S. Coluccini, *Secure Semi-Passive RFID Tags—Prototype and Analysis*, tech. report, Bridge Project, Nov. 2008; www.bridge-project.eu/data/File/BRIDGE_WP04_Secure_semi-passive_RFID_Tags.pdf.
2. A. Arbit, Y. Oren, and A. Wool, "Toward Practical Public Key Anti-counterfeiting for Low-Cost EPC Tags," *Proc. 2011 Int'l IEEE Conf. RFID* (RFID 11), 2011, pp. 184–191.
3. D.V. Bailey and A. Juels, "Shoehorning Security into the EPC Tag Standard," *Proc. 5th Int'l Conf. Security and Cryptography for Networks* (SCN 06), LNCS 4116, 2006, pp. 303–320.
4. M. Feldhofer, S. Dominikus, and J. Wolkerstorfer, "Strong Authentication for RFID Systems Using the AES Algorithm," *Proc. 6th Int'l Conf. Cryptographic Hardware and Embedded Systems* (CHES 04), LNCS 3156, Springer, 2004, pp. 357–370.
5. G. Gaubatz et al., "State of the Art in Ultra-Low Power Public Key Cryptography for Wireless Sensor Networks," *Proc.* 3rd *IEEE Int'l Conf. Pervasive Computing and Communications Workshops*, 2005, pp. 146–150.
6. H. Holtzman, S. Lee, and D. Shen, "Opentag: Privacy Protection for RFID," *IEEE Pervasive Computing*, vol. 8, no. 2, 2009, pp. 71–77.
7. *ISO/IEC 18000-6 Information Technology—Radio Frequency Identification for Item Management—Part 6: Parameters for Air Interface Communications at 860 MHz to 960 MHz*, ISO/IEC, 2011.
8. A. Juels and S. Weis, "Authenticating Pervasive Devices with Human Protocols," *Proc. Advances in Cryptology* (Crypto 2005), LNCS 3621, 2005, pp. 293–308.
9. D. Naccache, *Method, Sender Apparatus and Receiver Apparatus for Modulo Operation*, European Patent Application 91402958.2, filed 27 Oct. 1992.
10. Y. Oren and M. Feldhofer, "WIPR—Public-Key Identification on Two Grains of Sand," *Proc. Workshop on RFID Security 2008*, 2008, pp. 15–27.
11. Y. Oren and M. Feldhofer, "A Low-Resource Public-Key Identification Scheme for RFID Tags and Sensor Nodes," *Proc. 2nd Int'l Conf. Wireless*

the AUTHORS



Alex Arbit is a Hardware & Electronics Engineer at Tel Aviv University. His research interests include real-world cryptography and low-resource cryptographic constructions for lightweight computers. Arbit has an MSc in electrical engineering from Tel Aviv University. Contact him at alexand5@eng.tau.ac.il.

Yossef Oren is a post-doctoral research scholar in the Department of Computer Science at Columbia University. His research interests include power analysis attacks and countermeasures, low-resource cryptographic constructions for lightweight computers, and real-world cryptography. Oren has a PhD in electrical engineering from Tel Aviv University. Contact him at yos@cs.columbia.edu.

Avishai Wool is cofounder of the AlgoSec Systems (formerly Lumeta) network security company and is an associate professor at Tel Aviv University's School of Electrical Engineering. His research interests include firewall technology, computer, network, and wireless security, smart-card and RFID systems, and side-channel cryptanalysis. Wool has a PhD in computer science from the Weizmann Institute of Science, Israel. He is the creator of the AlgoSec Firewall Analyzer, a senior member of IEEE, and a member of the ACM and Usenix. Contact him at yash@eng.tau.ac.il.

Network Security (WiSec 09), ACM, 2009, pp. 59–68.

12. P. Rotter, "A Framework for Assessing RFID System Security and Privacy Risks," *IEEE Pervasive Computing*, vol. 7, no. 2, 2008, pp. 70–77.
13. A. Shamir, "Memory Efficient Variants of Public-Key Schemes for Smart Card Applications," *Proc. Advances in Cryptology* (EuroCrypt 94), LNCS 950, 1995, p. 445.
14. S.A. Weis et al., "Security and Privacy Aspects of Low-Cost Radio Frequency Identification Systems," *Security in Pervasive Computing* (SPC 2003), LNCS 2802, 2003, pp. 201–212.
15. J. Wu and D.R. Stinson, "How to Improve Security and Reduce Hardware Demands of the WIPR RFID Protocol," *Proc. 2009 IEEE Int'l Conf. RFID* (RFID 09), 2009, pp. 192–199.