Algebraic Eraser™: A lightweight, efficient asymmetric key agreement protocol for use in no-power, low-power, and IoT devices

Derek Atkins, SecureRF Corporation

The Algebraic Eraser™ (AE) provides a public-key Diffie-Hellman style key agreement protocol that performs significantly better than ECC in both software and hardware. One hardware implementation in 65nm CMOS performs 70-200 times better than ECC in speed and power usage. Software implementations also perform 50-200 times faster than ECC (for example on an MSP430 using around 5000 bytes of code and 500 bytes of RAM). Moreover, the basic algorithm under AE, called E-Multiplication™, can also be used to create a Hash, Block Cipher, PRNG, Stream Cipher, and Signature algorithm. This presentation introduces the Algebraic Eraser algorithm, the underlying math, and shows how it can be applied to real-world uses.

SecureRF Corporation
100 Beard Sawmill Road
Suite 350
Shelton, CT 06484

203-227-3151
info@SecureRF.com
www.SecureRF.com

1. Introduction

As ubiquitous as the Internet is today, it was not until the 1990's that the necessary security protocols were put in place that enabled users to function securely while online. For example, the online revolution in the financial sector, e-commerce, and medical records, all rely on the security breakthrough provided by public-key cryptography. Public-key cryptosystems fall into two categories: the Diffie-Hellman (DH) protocol published by Whitfield Diffie and Martin Hellman in 1976, and the RSA protocol, as publicly described by Ron Rivest, Adi Shamir, and Leonard Adleman in 1978. The deployments of RSA, DH, and a more recent Diffie-Hellman like method, Elliptic Curve Cryptography (ECC), have brought solutions to the current Internet.

In many respects we are at a similar evolutionary point with devices found in the Internet of Things (IoT) that are used for connectivity, credentials, the Smart Grid and industrial controls. The almost 40 year old solutions that enabled the Internet revolution are not suitable for microcontrollers; wireless sensors and devices utilizing Near Field Communication (NFC), radio frequency identification (RFID) or Bluetooth Low Energy (BLE); embedded systems and other low-resource environments being implemented as part of the Internet of Things; a new generation of public-key infrastructure needs to emerge.

At issue with today's commonly implemented RSA and Diffie-Hellman type public-key protocols, including ECC, concerns the computational footprint they entail. While memory and energy usage is not a primary concern for most environments requiring cryptographic security, these issues, along with runtime, lie at the heart of any small computing device security discussion. Every one of these cryptographic systems at its core utilizes multiplication of large numbers. As a result the computing resources required to achieve security grow rapidly as the level of security is increased. Even ECC, with its lower resource usage than RSA or DH, provides inadequate performance in constrained devices, to the point where developers are not even considering public key solutions.

The Algebraic Eraser™ (AE), introduced by Anshel, Anshel, Goldfeld, and Lemieux in 2006, is a Diffie-Hellman-like key agreement protocol which was designed to be suitable for implementation on low-cost platforms with constrained computational resources which are associated with the Internet of Things. It addresses the privacy and security needs of the IoT-related consumer products, medical devices, building/home automation, credentials, automotive systems, and mobile payments now being developed.

2. Enter the Algebraic Eraser

The foundation of the Algebraic Eraser public-key cryptosystem together with its security lie in three distinct areas of mathematics: the theory of braids, the theory of matrices with polynomial entries (expressions of finite length constructed from variables), and modular arithmetic in small finite fields. At its core is a highly specialized function (replacing the standard system's operations), known as E-Multiplication™, which brings together these mathematical tools and enables the system to provide high-speed security without overwhelming the memory and power available. This core function is highly resistant to reverse engineering due to the connections with mathematically intractable problems arising from the symbiotic combination of these disparate fields of mathematics..

Security protocols which are based on secret methods are often insecure because they have not had the benefit of wide spread testing and analysis. Strong cryptographic security methods are published for peer review. Details about how the Algebraic Eraser's key agreement protocol for public key cryptography is suitable for low resource devices, such as RFID tags, have been published by The American Mathematical Society in the peer-reviewed book "Algebraic Methods in Cryptography" in the section titled *Key agreement, the Algebraic Eraser™ and Lightweight Cryptography*. A version of this paper is available on SecureRF's

web site (http://www.securerf.com/wp-content/uploads/2014/03/SecureRF-Technical-White-Paper-06-with-Appendix-A-B.pdf). For convenience we include some of the mathematical description of AE in Section 5. The reader is encouraged to also refer to the original AE paper for a deeper description.

The Algebraic Eraser key agreement protocol (AEKAP, also called AEDH), when viewed at a high level, brings together the Braid group and a rapidly computable Algebraic Eraser based irreversible (one-way) function whose output consists of ordered pairs of specialized matrices and permutations. In the AEKAP each user is equipped with a collection of braid group elements from which the Braid element part of the user's private key is constructed. Once the private key is specified the user's public key is evaluated using the above mentioned one-way functions to produce a matrix/permutation ordered pair. To execute AEDH the users exchange their respective public keys (over an insecure channel) and each user, by combining the received public key and their own private key can evaluate, via the same one-way function, the unique common key (which is yet again a matrix/permutation pair) that is the output of the AEKAP.

The security of the AEKAP rests on the infeasibility of reversing the Algebraic Eraser based one-way function utilized in the protocol together with the difficulty of solving certain simultaneous equations over the braid group, aka the *Simultaneous Conjugacy Separation Search Problem*.

Among the unique features of the protocol is the fact that the protocol scales linearly in execution time. For example if there is a need to increase the security level, e.g. from 128 to 256 bits, the protocol execution time would scale linearly (i.e. double) and the size of both the public and exchanged keys would not necessarily need to increase. In tests the performance is actually slightly better that this, showing an increase of a factor of 1.9 instead of 2, whereas the performance penalty of ECC in the same increase adds a factor of 4.5 to the execution time.

The process of producing the user collections of braid element, which is done off-line, insures the Braid based security feature of the AEKAP is in place. There is no possible Braid based attack on the public-key due to the fact the public keys are not themselves braids but are the outputs of the one-way function applied to braids, resulting in a matrix and permutation.

There are several groups working on mathematical security using braids. The single conjugate work of Ko-Lee, et al from Korea is a substantially different method then SecureRF's multiple conjugate protocol. The security for the Ko-Lee et al Braid based key agreement protocol (KL…KAP) is based on a special case of solving a single equation in the Braid group. Each user chooses a private key, which is a braid element that commutes with the other user's private key. Each user's public key is obtained by applying a function to the private key whose output is another Braid group element. Within a more general context this function would be difficult to reverse, but the fact the inputs commute impacts the situation significantly. A specialized Braid based attack can be used and no further one-way functions are there to thwart the attack. One other contrasting point is that the size of the public key and the exchanged key grows as the size of the private key increases and the running time of the Ko-Lee et al scheme is quadratic in the length of the private key compared to a linear run time for the AEKAP. Thus a requirement of higher security would make for larger blocks of data to be transmitted and a more intensive computation would be required to obtain the exchanged key. In a resource-constrained environment this is a constant concern. The published attacks on the Ko Lee, et al. single conjugate method inherently do not work in the multiple-conjugate methods used in the Algebraic Eraser protocol. To date, all published attacks on the Algebraic Eraser protocol have been refuted, c.f. *Defeating the Kalka--Teicher--Tsaban linear algebra attack on the Algebraic Eraser* (http://arxiv.org/pdf/1202.0598v1) and *On the cryptanalysis of the generalized simultaneous conjugacy search problem and the security of the Algebraic Eraser* (http://arxiv.org/pdf/1105.1141v1).

2.1. Algebraic Eraser Design

E-Multiplication is the core foundation of AEDH, but it can be used to derive other cryptographic primitives as well. Using the same E-Multiplication core engine one could create a Hash, a Block Cipher, a Pseudo-Random Number Generator, a Stream Cipher, and even a Digital Signature Algorithm. The benefits of having all these primitives based around a single core is evident in a hardware implementation, where you can re-use the same silicon footprint across each primitive, thereby reducing significantly the number of gates required to implement the whole suite of algorithms.

The key agreement protocol is the oldest of the AE methods. It splits the braid group into two halves to create sets of commuting conjugates that get randomly combined to form private keys. The participants use E-multiplication over the series of conjugates to compute the shared secret. However using the same E-multiplication one can, for example, compute a Hash. More specifically, an AE Hash algorithm is a family of hashes where you start with an initial matrix and a set of hash braids and then break the message down into small blocks (of 3-8 bits) and apply the appropriate hash braid for the block. If you change the set of hash braids (even just permuting the set) you can achieve a "new" hash function. This is why it's a "family."

E-multiplication is also at the core of other cryptographic primitives, including a block cipher, pseudo-random number generator, stream cipher, and signature algorithm. However we focus on the key agreement protocol here.

2.2. Quantum Resistance

While not exhaustively tested against every know quantum attack method, the Algebraic Eraser does not appear to be susceptible to Shor's quantum algorithm. Further testing is desired.

3. Implementation Experiences

The Algebraic Eraser Key Agreement Protocol has been implemented in Software (C, Java, and platform-specific Assembly) and in Hardware (VHDL and Verilog). The following sections detail several implementations of AEDH and how it compares to ECC.

3.1. TI MSP430

The Texas Instruments (TI) MSP430 microcontroller is used extensively for sensors and low power (or even passive) devices. Running at 16MHz the implementation of AE required just under 4KB of ROM and 544 bytes of RAM, running in under 50mS (versus an estimated ECC execution of 1.5 seconds). This was at a security level of 2^{80}using lookup tables and written in assembly language. An implementation in pure C required 65ms to complete a shared secret calculation, requiring only 3440 byte of code and 544 bytes of RAM.

We can increase the security level to 2^{128} and use a log/antilog tables for our field computation. In that C implementation, taking 2359 bytes of ROM and 585 bytes of RAM, a shared secret computation required 106ms at 16MHz.

Notice the linear nature of the Algebraic Eraser. Increasing the security level from 80 to 128 bits (a 60% increase) required an increase in execution time from 65ms to 106ms (63%). Compare that to the increase in ECC execution time when you increase the security level.

Others have researched ECC on the MSP430; In particular *Softtware implementation of Pairing-Based Cryptography on sensor networks using the MSP430 micro controller*, by C.P.L. Gouvêa and J. López (published in Progress in Cryptology –- Indocrypt 2009) shows results that required upwards of 20-30KB of ROM, 2-5KB of RAM, and required anywhere from 600 to 1500ms to compute a shared secret.

Security Level	Algorithm	Language	ROM	RAM	Speed
80	**AE**	**C**	**3440**	**544**	**65ms**
80	**AE**	**C + Assembly**	**3990**	**544**	**50ms**
80	ECC	C	23,000	2500	600ms
128	**AE**	**C**	**2359**	**585**	**106ms**
128	ECC	C	25,000	3500	1500ms

Table 1: Comparing Algebraic Eraser to ECC on a TI MSP430 at 16MHz

3.2. ARM Cortex M3

The ARM family of processors is probably the most widely used 32-bit CPUs in the IoT space. While the ARM Cortex M0 is the smallest of the set, we focused on the ARM Cortex M3 (using an mbed LPC1768), a very popular choice and easily available for development.

While on the TI MSP430 is initially focused on a security level of 2^{80}, on ARM we skipped that and moved directly to the 2^{128} security level. Like the MSP430 we used log/antilog tables to compute the field computations. Using an implementation in a combination of C and assembly language we measured 14.9ms to compute a shared secret using 2065 bytes of ROM and 544 bytes of RAM at 48MHz. A C only implementation took a bit longer, 34ms, using 3339 bytes of ROM and 521 bytes of RAM.

One of the reasons Algebraic Eraser runs so efficiently is the small sizes of the fields used. For example, at the 128-bit security level our field is literally 256 (2^8). This actually is a detriment to AEDH because the ARM is a 32-bit processor with a 32-bit data bus. This means that without additional improvements we're only using 25% of the available system. However, because the Cortext M3 has a 32x32→64 bit multiplier, we can increase the security level of AEDH and, at the same time, use the ARM processor more fully.

To that end we changed the parameters used in AEDH to use a braid with fewer strands and extended to field to use the Mersenne Prime 2^{31}-1 (which we call M31) and results in a security level of 2^{310}. Because of the nature of working with M31, an implementation in C took 656 bytes of ROM, ran in 820 bytes of RAM, and computed a shared secret in 74ms at 48MHz.

Again, we see the linear nature of the Algebraic Eraser. Increasing the security level from 128-bit to 310-bit (a 142% improvement) required a 118% runtime increase, from 34ms to 74ms.

Now let us compare AE to ECC. Wenger, Unterluggauer, and Werner in *8/16/32 Shades of Elliptic Curve Cryptography on Embedded Processors* in Progress in Cryptology, Indocrypt 2013, showed a full assembly-language implementation of ECC in 7168 bytes of ROM that required 540 bytes of RAM but still required 233ms to compute a shared secret. We also created an implementation using WolfSSL in C; this used 9780 bytes of ROM, 7456 bytes of RAM, and required 889ms to compute a shared secret. Moreover, employees at ARM reported at the IETF-92 in March 2015 (H. Tschofenig, M. Pégourié-Gonnard, *Crypto Performance on ARM Cortex-M Processors*) that they measured ECDHE on the LPC1768 in 864ms.

Security Level	Algorithm	Language	ROM	RAM	Speed
128	**AE**	**C + Assembly**	**2065**	**544**	**15ms**
128	**AE**	**C**	**3339**	**521**	**34ms**
128	ECC	Assembly (M0)	7168	540	233ms
128	ECC	C (ARM)	(unavailable)	(unavailable)	864ms
128	ECC	C (WolfSSL)	9780	7456	889ms
310	**AE**	**C**	**656**	**820**	**74ms**

Table 2: Comparing Algebraic Eraser to ECC on an ARM Cortex M3 at 48MHz

3.3. 65nm CMOS

More impressive is the improvement of AEDH over ECDH in hardware. In 65nm CMOS the Algebraic Eraser requires 20,206 gates and runs in 3,352 cycles. ECC, on the other hand, while implemented in 29,458 gates takes a whopping 164,823 cycles (c.f. *A Flexible Soft IP Core for Standard Implementations of Elliptic Curve Cryptography in Hardware*, B. Ferreira and N. Calazans, 2013 IEEE 20th International Conference on Electronics, Circuits, and Systems (ICECS), 12/2013). An ECC engineer will continue to explain that they can improve the performance by adding parallelisms to the implementation, which is true. Increase the gate count to 77,858 (a factor of 2.6) and the clock count reduces to 85,367 cycles (a 48% improvement). Adding another 2.5x to 195,382 gates increases the speed another 13% to 70,469 cycles. The diminishing returns of each further "optimization" of ECC caused the writers to stop there.

Recall that the number of gates in a circuit is directly proportional to the amount of power it requires. In order to better compare the multiple optimizations of ECC to AE one can look at the product of cycles and gates, showing effectively the "speed times power" of that configuration. This allows a more direct method to compare, as shown in Table 1:

ECC			AE			Gain
Cycles	Gates	Weighted Performance	Cycles	Gates	Weighted Performance	Speed & Power
164,823	29,458	4,855,355,934	3,352	20,206	67,730,512	71.7x
85,367	77,858	6,646,503,866				98.1x
70,469	195,382	13,768,374,158				203.3x

Table 3: Comparing Algebraic Eraser to ECC in 65nm CMOS

This table shows clearly that as the additional optimizations are added to ECC the overall speed times power gains of AE over that configuration actually increase. This is due to the non-linearity of the ECC improvements.

4. Algebraic Eraser Standards Initiatives

SecureRF is active in numerous standards organizations. In ISO the Algebraic Eraser is currently a Working Draft in JTC-1/SC-31/WG-7 as item 29167-20, and also expected in JTC-1/SC-27/WG-2 (although it has not, yet, been submitted as a New Work Item). In the IETF AE is currently proposed as an extension to

RFC4880, and additional submissions will follow documenting how to use AE in an X509 certificate and how to use AEDH in TLS. Discussion is also currently ongoing in the CFRG.

5. Algebraic Eraser Math in a Nutshell

The AE Key Exchange Protocol (AEKAP) enables two users, Alice and Bob, to evaluate a shared secret using their own private key and the public key of the other user. The following definitions are used in the description algorithm which provides the security for this protocol.

The AEKAP contains the following public information:

- A fixed matrix $M_0 \in GL(F_q)$, which is chosen via a proprietary method,
- A set of conjugates in B_N for each user, Alice:

$$\{c_1 = z\, a_1\, z^{-1}, c_2 = z\, a_2 z^{-1}, \ldots, c_k = z\, a_k z^{-1}\} \quad ,$$

  and Bob:

$$\{d_1 = z\, b_1\, z^{-1}, d_2 = z\, b_2 z^{-1}, \ldots, d_l = z\, b_l z^{-1}\} \quad ,$$

  where it is assumed that each of the conjugates is *rewritten*, i.e., a Braid group algorithm such as Dehornoy or Birman-Ko-Lee is applied to the conjugates making the element z intractable to derive. The Braid element z, together with the Braid group elements $a_1, \ldots, a_k, b_1, \ldots, b_l$ are chosen via a proprietary method to insure security. One important feature of these sets of braid elements is that $a_i b_j = b_j a_i$, for all $i = 1, \ldots, k, j = 1, \ldots, l.$

- A set of Tvalues, which is an array of N entries in F_q that get used as part of the E-multiplication.

This public information (matrix, user conjugate set, and Tvalues) make up the keyspace for AEKAP. For two users to communicate they must share a common keyspace. This is similar to Diffie-Hellman where you choose a common prime, or in ECC where you choose a common curve. In AEKAP you choose a common matrix and conjugate set. The main difference in that there are two sets of conjugates in the set and each user must choose theirs from the opposite set (e.g., tags choose from set A, and readers from set B).

Note that M_0 must be of a special form to prevent certain classes of weak-key attacks, similar conceptually to the attacks possible if an RSA key is chosen randomly or chosen as the product of two poorly chosen large random primes instead of using two large primes with properties that defeat known attacks. The special method of choosing the matrix assures that this class of attack is not possible. Similarly, braid element z must be chosen to be large enough to prevent a different class of attacks, similar conceptually to choosing a Diffie-Hellman prime that's too small.

The user private/public key pairs of the AEKAP: User Private keys have two components

- The 1st Private key is a polynomial of degree N-1 in the matrix M_0 with coefficients in the field F_q : in the case of Alice,

$$M_A = \sum_{i=0}^{N-1} \alpha_i M_0^i.$$

In other words, the 1st Private Key (M_A) is an *NxN* Matrix where each of the N^2 entries is a member of the field F_q -- computed based on the public keyspace matrix M_0 .

– The 2nd Private key consists of a product of a sequence of the user's conjugates, again in the case of Alice,

$$c_{i_1} \cdot c_{i_2} \cdot \cdots \cdot c_{i_{L_A}}$$

which is itself an element in the Braid group.

In other words the user randomly chooses L_A conjugates (and their inverses) from the user's conjugate set in the keyspace and combines them together. This combination can happen in real time, or, because the result is just another entry in the Braid group (e.g. another conjugate) it can be reduced for storage, generally ending up about twice the size of the published conjugates.

– Alice's public key is obtained via e-multiplication:

$$\left(M_A, 1\right) \star \left(c_{i_1} \cdot c_{i_2} \cdots c_{i_{L_A}}, \sigma_{c_{i_1}} \cdot \sigma_{c_{i_2}} \cdots \sigma_{c_{i_{L_A}}}\right) \quad ,$$

where 1 is the identity permutation in S_N .

In other words, the result of the E-Multiplcation is a pair: an *NxN* matrix where each entry is a member of the field F_q , and a permutation of N entries (S_N). This is the composition of the Public Key.

Obtaining the shared secret/exchanged key:

– Alice receives Bob's public key,

$$\left(M_B, 1\right) \star \left(d_{j_1} \cdot d_{j_2} \cdot \cdots \cdot d_{j_{L_B}}, \sigma_{d_{j_1}} \cdot \sigma_{d_{j_2}} \cdot \cdots \cdot \sigma_{d_{j_{L_B}}}\right)$$

and computes

$$\left(M_A, 1\right) \cdot \left(M_B, 1\right) \star \left(d_{j_1} \cdot d_{j_2} \cdot \cdots \cdot d_{j_{L_B}}, \sigma_{d_{j_1}} \cdot \sigma_{d_{j_2}} \cdot \cdots \cdot \sigma_{d_{j_{L_B}}}\right) \star \left(c_{i_1} \cdot c_{i_2} \cdot \cdots \cdot c_{i_{L_A}}, \sigma_{c_{i_1}} \cdot \sigma_{c_{i_2}} \cdot \cdots \cdot \sigma_{c_{i_{L_A}}}\right).$$

Likewise, Bob receives Alice's public key and computes

$$\left(M_B, 1\right) \cdot \left(M_A, 1\right) \star \left(c_{i_1} \cdot c_{i_2} \cdot \cdots \cdot c_{i_{L_A}}, \sigma_{d_{i_1}} \cdot \sigma_{d_{i_2}} \cdot \cdots \cdot \sigma_{d_{i_{L_B}}}\right) \star \left(d_{j_1} \cdot d_{j_2} \cdot \cdots \cdot d_{j_{L_B}}, \sigma_{d_{j_1}} \cdot \sigma_{d_{j_2}} \cdot \cdots \cdot \sigma_{d_{j_{L_B}}}\right).$$

Both of these computations result in the shared secret/exchanged key:

$$\left(M_A, 1\right) \cdot \left(M_B, 1\right) \star \left(d_{j_1} \cdot d_{j_2} \cdot \cdots \cdot d_{j_{L_B}}, \sigma_{d_{j_1}} \cdot \sigma_{d_{j_2}} \cdot \cdots \cdot \sigma_{d_{j_{L_B}}}\right) \star \left(c_{i_1} \cdot c_{i_2} \cdot \cdots \cdot c_{i_{L_A}}, \sigma_{c_{i_1}} \cdot \sigma_{c_{i_2}} \cdot \cdots \cdot \sigma_{c_{i_{L_A}}}\right) = \left(M_B, 1\right) \cdot \left(M_A, 1\right) \star \left(c_{i_1} \cdot c_{i_2} \cdot \cdots \cdot c_{i_{L_A}}, \sigma_{d_{i_1}} \cdot \sigma_{d_{i_2}} \cdot \cdots \cdot \sigma_{d_{i_{L_B}}}\right) \star \left(d_{j_1} \cdot d_{j_2} \cdot \cdots \cdot d_{j_{L_B}}, \sigma_{d_{j_1}} \cdot \sigma_{d_{j_2}} \cdot \cdots \cdot \sigma_{d_{j_{L_B}}}\right).$$

As before, the result of this computation is an *NxN* Matrix and a Permutation.

6. About SecureRF

The inventors of SecureRF's public-key cryptosystem are co-founders Dr. Michael Anshel, Dr. Dorian Goldfeld and Dr. Iris Anshel.

Dr. Michael Anshel is a security thought-leader and world-class mathematician with expertise in the field of cryptography. Dr. Anshel has authored and co-authored numerous papers in the area of public-key cryptography, is the co-inventor of four patents in the area of cryptography, zeta-one-way functions, and braid group and has received numerous fellowships and honors. He is a Professor Emeritus in the Department of Computer Science at The City College of New York.

Dr. Goldfeld is a world-class mathematician who has published over 100 papers and books and lectured internationally on a wide range of cryptographic topics and methods including applications of elliptic curves, quadratic fields, zeta functions, public-key cryptography, and group theoretic approaches to public-key cryptography. In 2009 he was inducted as a Fellow of the prestigious American Academy of Arts & Sciences. He is the co-inventor of five patents in the areas of multistream encryption systems, high speed cryptography, cryptographically secure algebraic key establishment protocols based on monoids, and cryptographic hash functions. He has been a professor in the Faculty of Mathematics at Columbia University since 1985.

Dr. Iris Anshel, SecureRF's Chief Scientist, is an accomplished mathematician and cryptographer. In addition to extensive research and publications, Dr. Anshel has experience in the commercialization of security technology. As a co-founder of Arithmetica Inc she was responsible for documenting methods for commercial deployment of new cryptography protocols including the AAG Braid Group Cryptosystem and supported sales and business development activity. She is co-inventor on three patents in the areas of cryptographically secure algebraic key establishment protocols based on monoids, and cryptographic hash functions.

Derek Atkins is the Chief Technology Officer at SecureRF, bringing over two decades of security protocol and implementation experience to the team. He has been a significant influence in the Internet Engineering Task Force (IETF), chairing several working groups and participating in multiple directorates including the Security Area Directorate. His software engineering experience includes numerous open- and closed-source projects including an implementation of the Pretty Good Privacy (PGP) protocol that has continued to last for over 15 years across five different company ownerships. His experience and focus ensures SecureRF delivers the optimum experience to its users and customers in both speed and security.

SecureRF Corporation – Securing the Internet of Things® – provides security solutions for embedded systems and wireless sensor technologies used in non-traditional payment systems, secure supply chain management, cold chain management, and anti-counterfeiting applications in the pharmaceutical, fashion, spirits, defense, and homeland security sectors. The company's technology is based on a breakthrough in public-key cryptography that is computationally efficient, yet highly secure and available as a software development kit, Verilog/VHDL, or as a core for FPGAs and ASICs. SecureRF also offers the LIME Tag™ - a range of highly secure NFC, UHF and Bluetooth LE sensor tags along with its anti-counterfeiting solution – Veridify™.