# Hardware Implementation of Bluetooth Security

*Bluetooth can implement its security layer's key-generation mechanism and authentication in software or hardware. Software implementation usually satisfies user requirements, but in time-critical applications or processing-constrained devices, a hardware implementation is preferable.*

**Paraskevas Kitsos, Nicolas Sklavos, Kyriakos Papadomanolakis, and Odysseas Koufopavlou**
*University of Patras, Greece*

Security in pervasive computing is a complex issue that has been the subject of negative publicity in recent years due to poor implementations (such as the Wired Equivalent Privacy protocol used by IEEE 802.11). Many low-level protocols are not secure, and the use of more secure high-level protocols is limited by the processing capabilities of mobile devices. Bluetooth could enhance and extend pervasive applications because it is well suited to the power requirements of mobile applications (see the "Bluetooth" sidebar). Furthermore, it offers methods for generating keys, authenticating users, and encrypting data.

Most single-chip Bluetooth baseband implementations, which include a low-performance general-purpose processor, implement only the data encryption in hardware. However, in time-critical applications requiring a fast connection and in devices with processing constraints, implementing key generation and authentication in the hardware (rather than software) is also preferable. To improve performance in these applications, we use an efficient implementation of the Safer+ (Secure And Fast Encryption Routine) algorithm, which reduces the resource requirements.

## Bluetooth security

The Bluetooth *security layer* uses four key elements: a *Bluetooth device address*, two separate key types (*authentication* and *encryption*), and a *random number*.[1] The authentication key generates the encryption key during the authentication phase. A Bluetooth device produces the random number using a random[2] or pseudorandom[3] process. In the Bluetooth system, the security layer is one of the baseband layers, which the upper layers (such as the link manager) control. It includes key management, key-generation mechanisms, user and device authentication, and data encryption. The Bluetooth security architecture can secure all upper layers by enforcing the Bluetooth security policy or can be flexible enough to let those layers use their own security policies.

### Key management and the key-generation mechanism

To connect, users must enter their personal identification numbers (see Figure 1). Using the PIN, the link-key-generation function ($E_2$) generates the appropriate link key. The authentication function ($E_1$) uses this key to authenticate the identity of the connecting parties. When data encryption is required, the encryption-key-generation function ($E_3$) produces the encryption key using the same link key. The stream cipher function ($E_0$) uses the encryption key to encrypt or decrypt the sending or receiving data.

During a connection, a Bluetooth device produces several different types of 128-bit link keys. These keys handle all security transactions between two or more parties. Depending on the application type, the link keys can be *initialization*, *unit*, *combination*, or *master* keys. A link key's lifetime depends on whether it is a semipermanent or temporary key. A point-to-multipoint connection, which transmits the same information to several recipients, uses a master (temporary) key. All other link-key types are semipermanent. In addition to link keys, there is the encryption key. The encryption-key-generation process uses the current link key to produce the encryption key.

1536-1268/03/$17.00 © 2003 IEEE ■ *Published by the IEEE CS and IEEE Communications Society*

# Bluetooth

Bluetooth is a technology and standard, designed as a wireless-cable replacement to connect a wide range of devices. Unlike wireless LANs such as 802.11b, it was designed to be low power, operate over a short range, and support both data and voice services. It enables peer-to-peer communications among many types of handheld and mobile devices. Furthermore, it provides a conceptually simple communication model and lets these devices exchange information and work together to benefit the user.

Bluetooth operates in the 2.45-GHz ISM (Industrial, Scientific, Medical) band and can be used all over the world. The communication channel between Bluetooth devices is based on Frequency Hop Spread Spectrum (FHSS) and switches between 79 frequencies at a rate of 1,600 frequencies per second, moving in a pseudorandom fashion. Air interface traffic is also encrypted using a key between 8 and 128 bits. To further enhance security, applications using Bluetooth can provide their own additional encryption layer. This multilayered approach considerably reduces the possibility of undetected and undesirable intrusion[1] (although some Bluetooth security weaknesses exist[2]).

## REFERENCES


1. R. Barber, "Security in a Mobile World–Is Bluetooth the Answer?" *Computers & Security*, vol. 20, no. 5, July 2001, pp. 374–379.
2. M. Jakobsson and S. Wetzel, "Security Weaknesses in Bluetooth," *Progress in Cryptology* (CT-RSA 2001), LNCS, vol. 2020, Springer-Verlag, San Francisco, 2001, pp. 176–191.

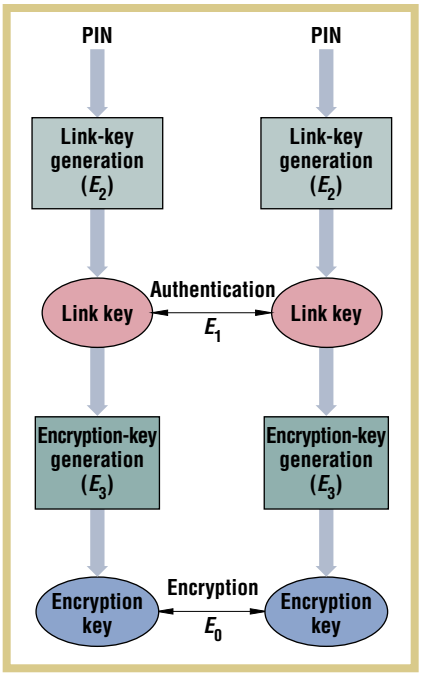




Figure 1. Key generation.

In the beginning of a connection (the initialization phase) between two parties, each party produces its own initialization key, $K_{INIT}$. The key-generation function $E_{22}$ (part of function $E_2$) generates this key, and its input parameters are the PIN code, the Bluetooth device address, and a 128-bit random number. $K_{INIT}$ produces the next link keys.

The two parties decide which link key (combination or unit key) to produce. When a device has limited memory, it chooses to produce a unit key. In this case, the device does not need to remember different keys for different links. Another key-generation function, $E_{21}$ (also part of function $E_2$), generates the unit key, and its input parameters are $K_{INIT}$, the Bluetooth device address, and a 128-bit random number. The two parties might also decide to use the combination key. In this case, in both devices, the key-generation function $E_{21}$ generates transient keys. The two devices exchange these keys and use them to produce the combination keys.

When a master device broadcasts data to several slave devices, it uses the master key. The master device generates the master key in the key-generation function $E_{22}$ using two 128-bit random numbers. The master device sends a random number to the slave devices. In all the devices (master and slave), the generation function $E_{22}$ uses this random number and the current link key to produce the overlay number. Then, the master device sends the master key bitwise XORed with the overlay to all the slaves so that each slave can calculate the master key.

### Authentication

Bluetooth authentication uses a challenge–response scheme, which checks whether the other party knows the link key. The scheme accomplishes authentication using the current link key. The process is symmetric, so a successful authentication requires that both parties share the same key.

First, the *verifier* (master) sends a random number $A$ to the *claimant* (slave) (see Figure 2). Then, both parties use the authentication function and input the random number $A$, the claimants' address (Bluetooth device address $B$), and the current link key, with length $L$, to produce the signed response (SRES). Then, the claimant sends the SRES to the verifier. Finally, the verifier checks if the two SRESs match.

However, the verifier isn't always the master; some applications require only one-way authentication. In both parties, the authentication function computes and stores the *authenticated ciphering offset* (ACO), generating the encryption key.

### Data encryption

When the link manager activates encryption, it generates the encryption key. The key-generation function generates the encryption key ($K_c$) from the current link key, a 96-bit *ciphering offset number*, and a 128-bit random number. The COF is based on the ACO, which is generated during authentication. The encryption key's size varies from 8 to 128 bits.

A Bluetooth system encrypts only the packet's payload, and it achieves this using the stream-cipher function ($E_0$). This function's input is $K_c$, the Bluetooth device address $A$, and some of the master real-time clock's data (see Figure 2). This func-

Figure 2. The authentication and encryption process.

tion consists of the *payload key generator*, the *key stream generator*, and the *encryption/decryption part*.[1]

The payload key generator combines the stream-cipher function's input bits in an appropriate order and leads them to the key stream generator's[4] four *linear feedback shift registers* (LFSR). The generator uses different encryption modes depending on the party link-key type. It doesn't encrypt the broadcast traffic when using a combination or unit key. However, when using a master key, it can use one of three possible encryption modes. In encryption mode 1, the generator doesn't encrypt anything. In encryption mode 2, it uses the master key to encrypt the individually addressed traffic but doesn't encrypt broadcast traffic. In encryption mode 3, it uses the master key to encrypt all traffic.

## Bluetooth implementations

Commercial chipsets are available that implement Bluetooth. Many companies (including Cambridge Silicon Radio, Philips, Ericsson, Atmel, and Zeevo) have integrated the basic Bluetooth baseband functions in hardware. Other companies, such as Silicon Wave, Oki Semiconductor, and Intel, provide a Host Control Interface that can be controlled, for example, across a USB interface. An ARM (Advanced Risk Machines) embedded processor core, for the appropriate firmware execution, supports the Bluetooth controller's hardware components (see Figure 3). We can implement the security protocol in these hardware components.

For an ARM-based system to work properly, it generally needs many RAM and ROM blocks, instruction cache memory, and sophisticated control logic. On the other hand, hardware implementations limit the silicon area and consume less of the chip's power. In addition, the software implementation requires multiply instructions and more clock cycles to execute the partial operation, reducing the system time performance.

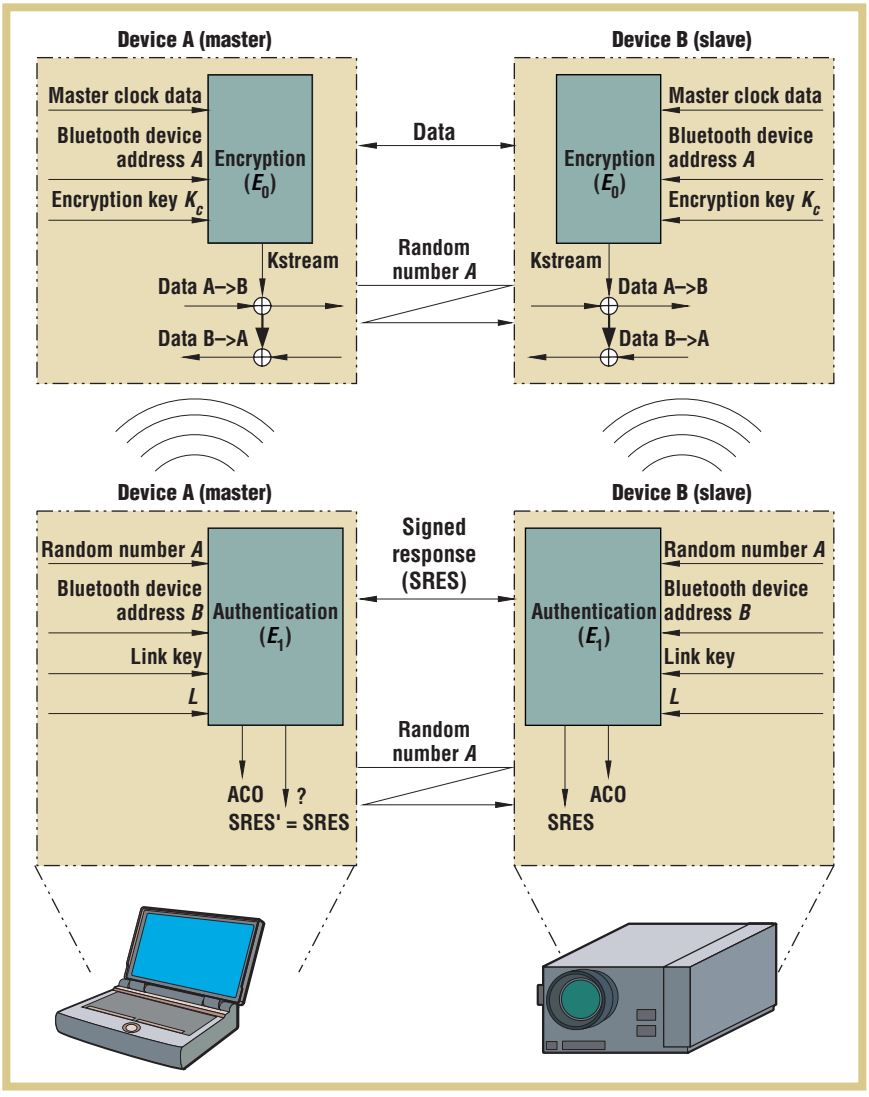
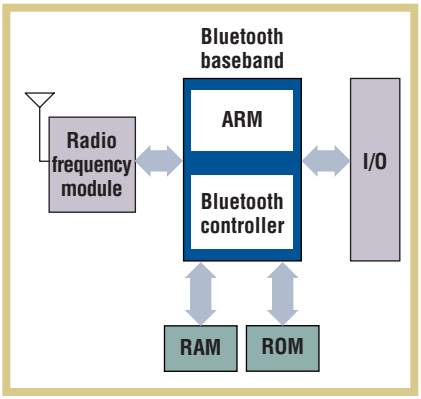



Figure 3. Bluetooth system hardware implementation.

The cryptography algorithms demand high processing capabilities. Implementing either a public-key encryption or a private-key algorithm, with a typical microcontroller instruction set, leads to large code size and high power requirements. The challenge is to include appropriate hardware primitives that make these operations more efficient in all these dimensions while letting the security algorithms evolve.[5]

## Security hardware implementation

As we stated earlier, most single-chip Bluetooth baseband implementations implement




data encryption (the stream cipher function unit) in hardware. This removes the continuous load from the processor across the wireless link during data transfer. In addition, implementing the key-generation

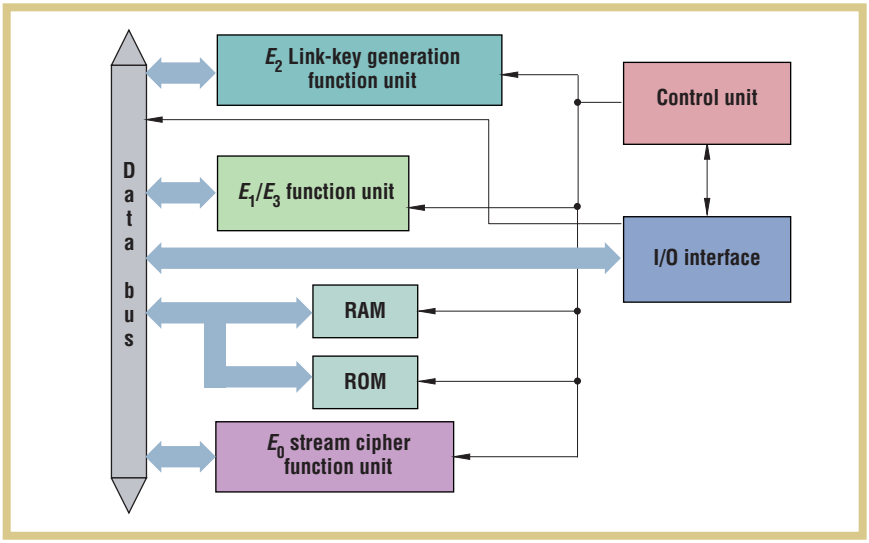



Figure 4. Security system architecture (the blue arrows are buses and the black arrows are control signals).

mechanism and authentication in hardware (rather than software) offers faster connection times and lower power consumption.

Figure 4 shows the proposed system architecture for the Bluetooth security hardware implementation. It consists of six units:

- The link-key-generation function
- The $E_1/E_3$ function (authentication and encryption-key generation)
- RAM and ROM memories
- The stream cipher function
- Control
- The I/O interface

The link-key-generation function unit produces the appropriate link keys. Using these link keys, the $E_1/E_3$ function unit performs authentication and produces the encryption key. It uses the stream cipher function unit for data encryption and transfers all the interunit data movements through the 64-bit data bus. It also integrates RAM to store the appropriate function keys and stores the function constants in the ROM. The control unit synchronizes the system's operation and puts the inactive system portions in standby mode.[6] This greatly reduces power dissipation. The system communicates with the external environment through the I/O interface.

The basic component of all the key generation functions ($E_2$, $E_3$) and the authentication function is the Safer+ algorithm.[1,7–9] Safer+ is based on the existing Safer family of ciphers, which comprises the ciphers Safer K-64, Safer K-128, and Safer SK-128,[7] developed by James Massey of ETH Zurich. All algorithms are byte-oriented block-encryption algorithms, which have two properties. First, to achieve the desired diffusion, they use a nonorthodox linear transformation (*Pseudo-Hadamard Transformation*). Second, they use additive constant factors (*bias vectors*) in the key scheduling to avoid weak keys.

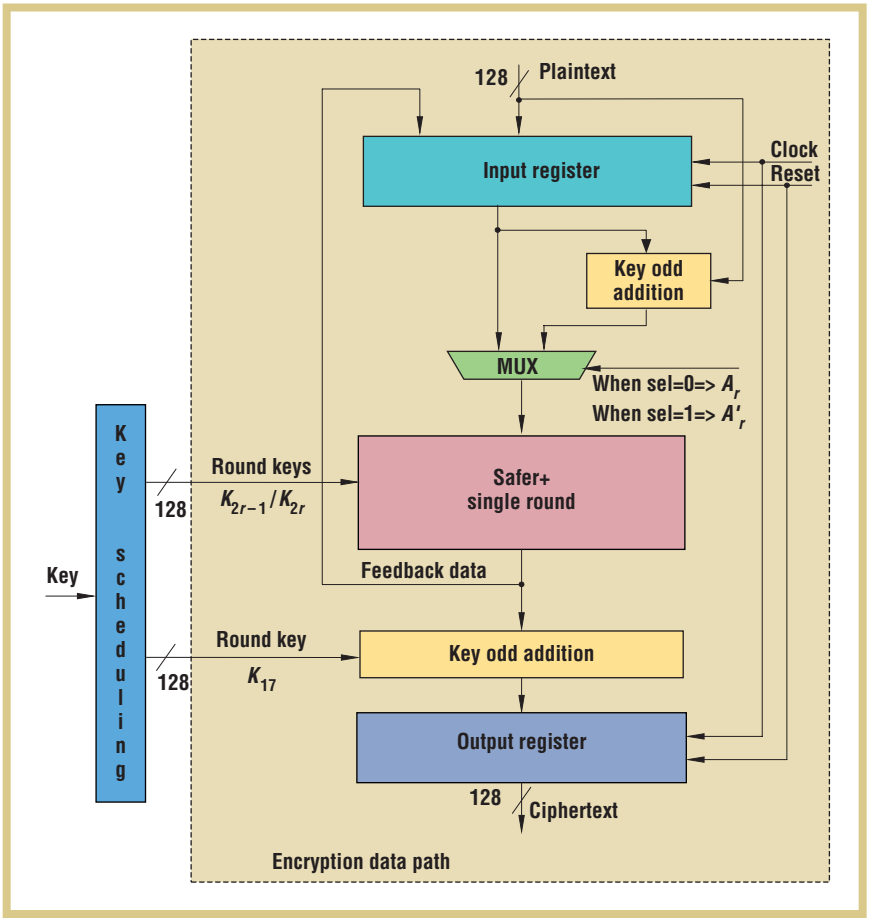



Figure 5. Safer+ implementation.

## Implementing the Safer+ algorithm

Figure 5 shows the Safer+ algorithm's hardware implementation. It consists of two main units: the encryption data path and the key-scheduling unit. The key-

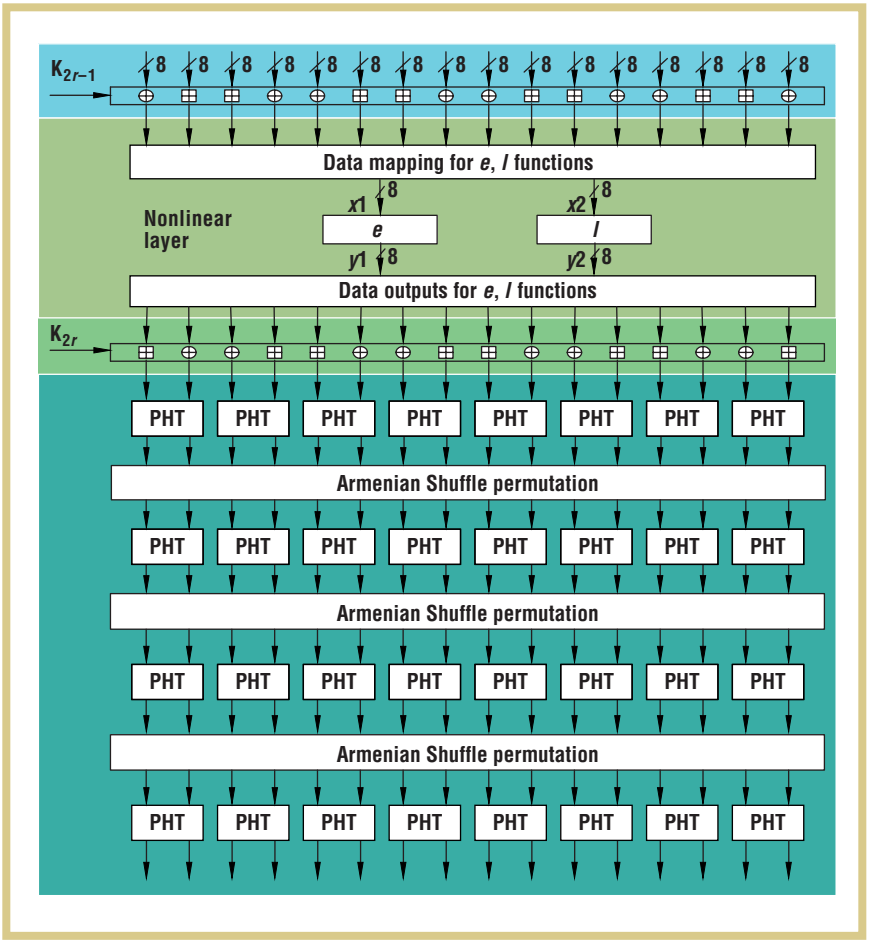



**Figure 6. A Safer+ single round (PHT stands for the Pseudo-Hadamard Transformation).**

scheduling unit allows on-the-fly computation of the round keys. To reduce the silicon area, we used eight loops of a key-scheduling single-round implementation. We implemented the bias vectors using 17- by 16-byte ROM blocks.

In the proposed design, round keys are applied in parallel in the encryption data path. The full Safer+ algorithm execution requires eight loops of the single round. We chose the single-round hardware implementation solution because, with this minimum silicon area, we could achieve the required throughput.

The encryption data path's first component is an *input register*, which combines the *plaintext* and the *feedback data* produced in the previous round. The input register feeds the Safer+ single round.

A Safer+ single round (see Figure 6) has four subunits:

- The mixed XOR/addition subunit, which combines data with the appropriate round subkey $K_{2r-1}$.
- The nonlinear layer (use of the nonlinear functions *e* and *l*). The *e* function is implemented as $y = 45^x$ in GF(257), except that $45^{128} = 0$. The *l* function is implemented as $y = \log_{45}(x)$ in GF(257), except that $\log_{45}(0) = 128$.
- The mixed addition/XOR subunit, which combines data with the round subkey $K_{2r}$.
- The four linear Pseudo-Hadamard Transformation layers, connected through an "Armenian Shuffle"[1,7–9] permutation.

We implement the nonlinear layer using a *data mapping* component that produces the X1 and X2 bytes. These bytes are the input of the nonlinear functions *e* and *l*. During one round, we execute *e* and *l* eight times. This design significantly reduces the required silicon area. Each function is implemented using 256 bytes of ROM.

After the Safer+ single round in the encryption data path is a mixed XOR/addition (or *key odd addition*) component. The encryption data path's final stage is the *output register* (throughout the rest of the article, we denote the Safer+ algorithm as function $A_r$). The authentication function uses the encryption functions $A_r$ and $A'_r$. There is only one difference between the two functions. In $A'_r$, the first round's input is added to the third round's input using the key-odd-addition subunit. $A_r$ and $A'_r$ are implemented as Figure 5 shows. The signal "Sel" determines the selection between $A_r$ or $A'_r$.

### The Link-key-generation function unit

Figure 7 illustrates the link-key-generation function unit. The $E_{21}$ and $E_{22}$ key-generation functions create the four authentication keys. The system implements these functions mainly using the Safer+ algorithm, grouping the four keys in pairs. It pairs the unit key with the combination key and the initialization key with the master key, storing them in 4- by 128-bit RAM.

### The $E_1$/$E_3$ function unit

The architectures of the authentication and encryption-key-generation functions are similar. So, we implemented these architectures in a reconfigurable unit (see Figure 8). The only difference is in their *expansion* subunit, which expands either the Bluetooth device address or the COF parameter into a 128-bit value. The *offset* module produces the key for the $A'_r$ function. The whole function needs 16 cycles to produce a new set of SRES and ACO parameters. If encryption is needed, the Link Manager sets the *Encryption_mode* signal so the unit operates as the encryption-key-generation function. After 16 cycles, this function creates the encryption key $K_c$.

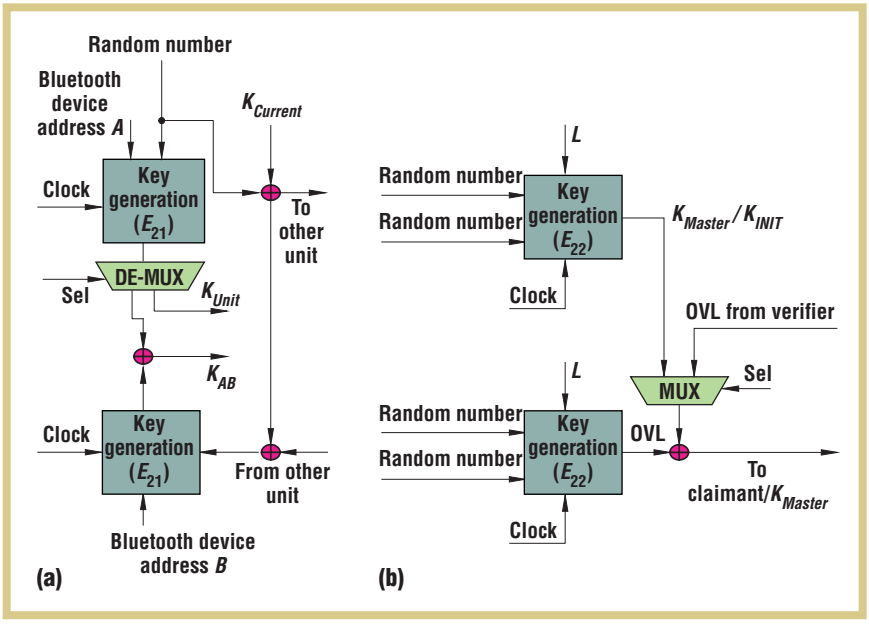



Figure 7. The link-key-generation function unit: unit ($K_{unit}$), combination ($K_{AB}$), initialization ($K_{init}$), and master key production ($K_{Master}$).

### The stream-cipher function unit

We need two components to implement the stream-cipher function unit (see Figure 9). The first is the *payload key generator* component, which has as input the encryption key $K_c$, the 48-bit Bluetooth device address, and the 26-bit master clock data. The second component (*key stream generator*) generates the key stream. It uses four LFSRs, whose output is the input of a 16-state finite-state machine. The state machine output is the key stream sequence.

The payload-key-generator component reduces the encryption key $K_c$ length and derives a new key $K'_c$. It uses the polynomial modulo operation $K'_c(x) = g_2(x)[K_c(x) \bmod g_1(x)]$, where $\deg(g_1(x)) = 8L$ and $\deg(g_2(x)) \leq 128 - 8L$. The Bluetooth system specifications specify the $g_1(x)$ and $g_2(x)$ polynomials. Figure 10 shows the proposed multiplier, which executes this operation. The coefficients of the $g_1(x)$ and $g_2(x)$ polynomials are stored in registers. The encryption key $K_c$ is shifted bit-by-bit, with the most significant bit first, in two $n$ and $m$ groups of serially connected D flip-flops. The $K_c$ reduction modulo $g_1(x)$ is executed, producing the *intermediate result* (IR(i)). Then, the IR(i) is multiplied by the $g_2(x)$ polynomial. The $g_1(x)$ and $g_2(x)$ degrees are $n$ and $m$, accordingly.

## Synthesis results and analysis

We captured the overall system (see Figure 4) using VHDL (VHSIC Hardware Description Language) and simulated/verified the system under real-time operating conditions. The performance, critical path, and silicon area of the Safer+ implementation determine the characteristics of the function implementations for link keys, encryption-key generation, and authentication.

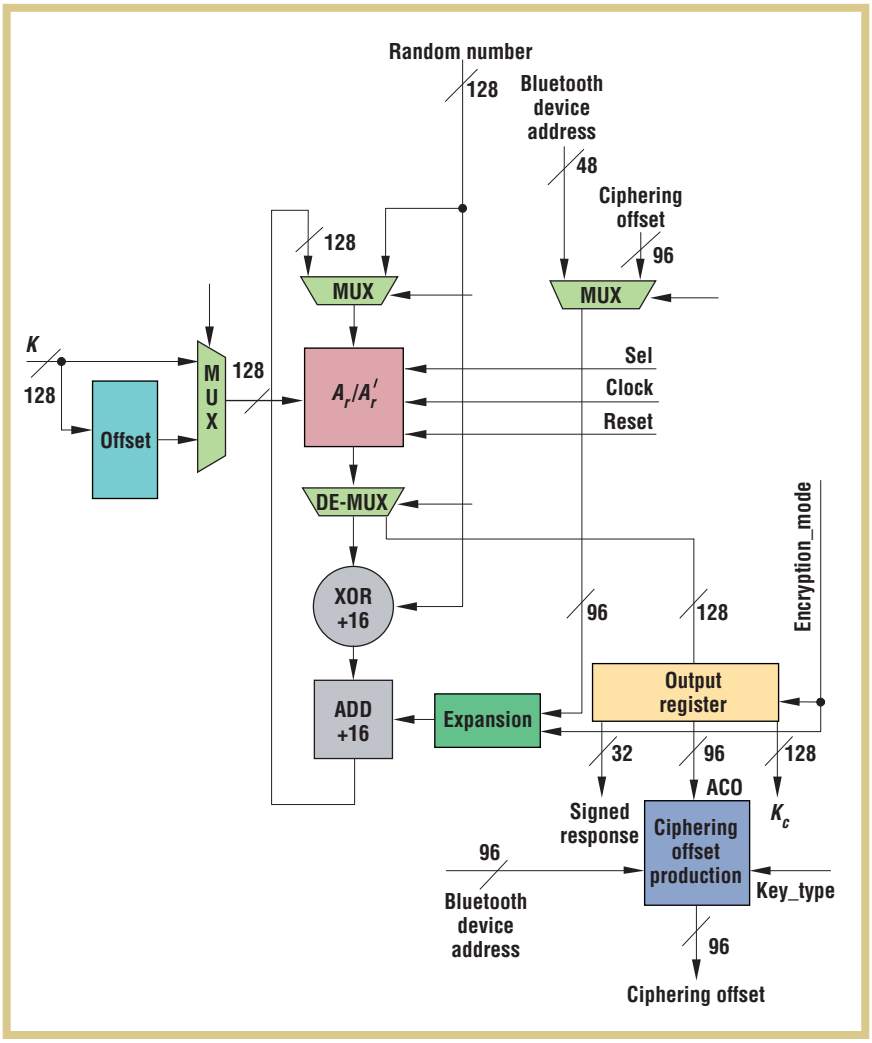
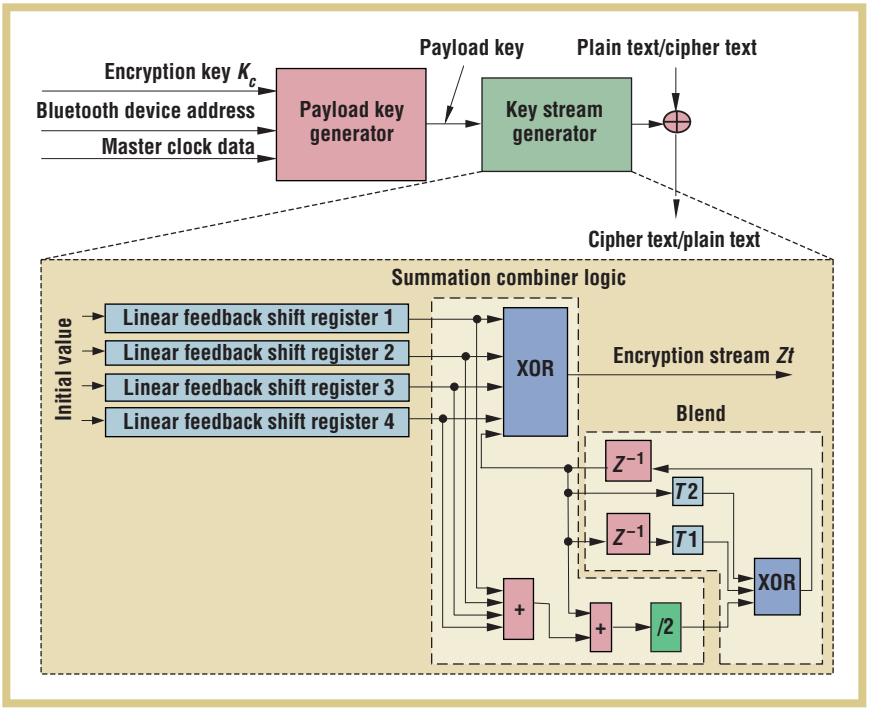



Figure 8. The $E_1/E_3$ (authentication and encryption-key generation) function unit.

Figure 9. The stream-cipher function unit.




Contrary to other implementations,[9,10] our Safer+ implementation includes only the encryption part. Bluetooth system requirements imposed this decision.[1]

The system performance of the proposed Safer+ algorithm implementation is very high, so the authentication or key-generation mechanism functions execute in less than 1.5 msec. Because our system uses only two nonlinear functions (*e* and *l*), it significantly reduces the silicon area.

The two components of the stream-cipher function unit are implemented easily using registers and basic logic gates (see Figures 9 and 10). The critical path of the key-stream generation is short, so the achieved throughput is much higher than the Bluetooth specification demands.

We synthesized, placed, and routed the system using the XILINX field-programmable gate array (VIRTEX-E V2600E-FG1156). Table 1 presents detailed synthesis results for the overall system. The device utilization for this specific XILINX FPGA device is 78.4 percent for the configurable logic blocks, 84 percent for the function generators, and 5 percent for the D flip-flops. The system clock frequency is 15 MHz. 80 bytes of RAM stores the keys that the link-key-generation and $E_1/E_3$ function units produce. The system also has a 4,704-byte ROM.

Figure 10. A proposed modulo multiplier.

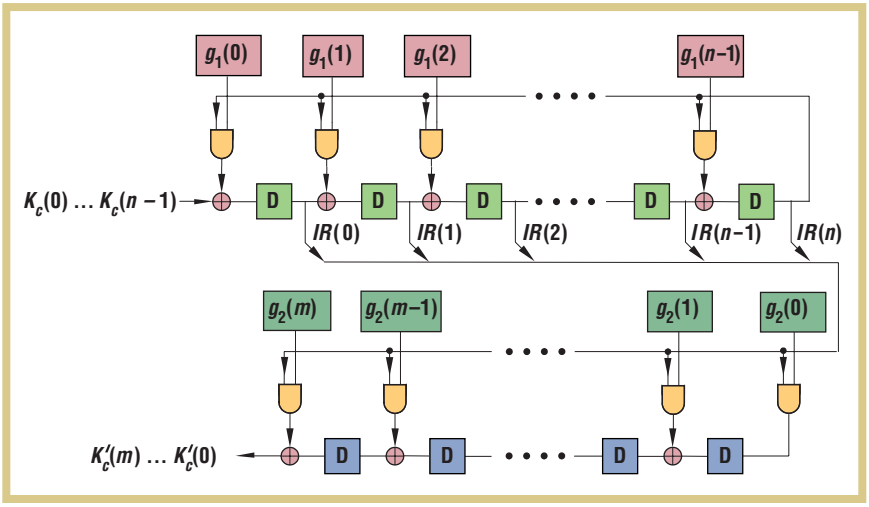



TABLE 1
System FPGA synthesis results.

| System component | Configurable logic blocks | Function generators | D flip-flops | ROM blocks (bytes) |
|---|---|---|---|---|
| Safer+ | 4,058 | 7,055 | 800 | 784 |
| $E_2$ link-key-generation function unit | 10,490 | 24,146 | 1,312 | 3,136 |
| $E_1/E_3$ function unit | 7,500 | 15,110 | 928 | 784 |
| $E_0$ stream cipher function unit | 895 | 1,508 | 300 | – |
| Control unit | 1,020 | 1,900 | 60 | – |
| Total system | 19,905 | 42,764 | 2,600 | 4,704 |

TABLE 2
Safer+ performance comparisons.

| Implementation | Frequency (MHz) | Throughput (Mbps) | Normalized frequency (MHz) |
|---|---|---|---|
| James Massey's first implementation[10] | 44 | 704.0 | 0.0625 |
| James Massey's second implementation[9] | 62 | 58.9 | 1.05 |
| Software1[10] | 200 | 33.0 | 6.06 |
| Software2[9] | 16 | 0.0256 | – |
| Our design | 20 | 320.0 | 0.0625 |

Table 2 presents the proposed Safer+ implementation's measurements, frequency, and throughput. For comparison, we also include measurements from other implementations. When establishing a Bluetooth connection using authentication and the key-generation mechanism, the throughput is much lower than when connecting using data encryption.

The Software1 implementation was coded in C language and executed on a 200-MHz Pentium PC.[10] The Software2 implementation was coded in Assembly language and executed on an 8-bit MCS 51 family processor.[9] The proposed Safer+ implementation's throughput is five times higher than James Massey's implementation,[9] and much higher when compared to the software implementations.

To fairly compare the Safer+ designs, Table 2 presents the normalized frequencies for each design for 1 million bits per second (Mbps) yielding throughput. The proposed implementation achieves the required throughput with the same clock frequency as one of Massey's designs.[10] Compared to previous designs, it requires a much lower clock frequency.

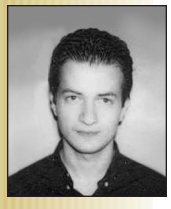
Pervasive computing promises a computing environment that seamlessly supports users in accomplishing their tasks and renders the actual computing devices and technology largely invisible. The explosive growth of both Internet and wireless communication means that technology can bring information, services, and applications to the user, anywhere. We might not be there yet, but we are headed in the right direction.

In realizing this vision, hardware and networking infrastructures are increasingly becoming a reality. Wireless networking standards, such as IEEE 802.11 and Bluetooth, provide local connectivity while the Internet provides worldwide connectivity. In particular, with Bluetooth, various devices can work together through ad hoc networks, marking the beginning of a standard that enables wireless machine-to-machine communications between each and every intelligent device and every appliance.

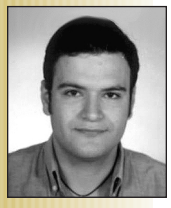
Over the last few years, we have labored under the constraint that secure cryptosystems require too much computation to be performed easily. Moore's law is producing

## the AUTHORS

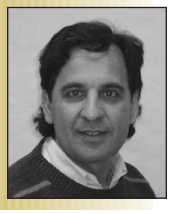
**Paraskevas Kitsos** is working toward his PhD in the Department of Electrical and Computer Engineering at the University of Patras. His research interests include Very Large Scale Integration design, hardware implementations of cryptography algorithms, security protocols for wireless communication systems, and Galois field arithmetic implementations. He received his BS in physics from the University of Patras. Contact him at VLSI Design Laboratory, Dept. of Electrical and Computer Eng., Univ. of Patras, 26500 Rio, Patras, Greece; pkitsos@ee.upatras.gr; www.vlsi.ee.upatras.gr/~pkitsos.



**Nicolas Sklavos** is pursuing his PhD in the department of Electrical and Computer Engineering at the University of Patras. His research includes Very Large Scale Integration and low-power design, cryptography implementations for wireless communications, and reconfigurable computing architectures. He received his diploma in electrical and computer engineering from the University of Patras. Contact him at the VLSI Design Laboratory, Dept. of Electrical and Computer Eng., Univ. of Patras, 26500 Rio, Patras, Greece; nsklavos@ee.upatras.gr; www.vlsi.ee.upatras.gr/~sklavos/sklavos.htm.



**Kyriakos Papadomanolakis** is pursuing his PhD in the department of Electrical and Computer Engineering at the University of Patras. His research interests include Very Large Scale Integration, low-power design, and fault-tolerant techniques. He received his diploma in electrical and computer engineering from the University of Patras. Contact him at the VLSI Design Laboratory, Dept. of Electrical and Computer Eng., Univ. of Patras, 26500 Rio, Patras, Greece; papk@ee.upatras.gr.



**Odysseas Koufopavlou** is an associate professor with the Department of Electrical and Computer Engineering at the University of Patras. His research interests include Very Large Scale Integration, low-power design, VLSI cryptosystems, and high-performance communication subsystems architecture and implementation. He received his Diploma and PhD in electrical engineering from the University of Patras. He is a member of the IEEE. Contact him at the VLSI Design Laboratory, Dept. of Electrical and Computer Eng., Univ. of Patras, 26500 Rio, Patras, Greece; odysseas@ee.upatras.gr; www.vlsi.ee.upatras.gr/~odysseas.

general-purpose processors that can handle the necessary crypto functions in a negligible fraction of their capacity. In recent years, we have also been able to build custom circuits that inexpensively fulfill the crypto demands of specialized applications. However, the cost and power requirements of high-performance general-purpose processors still make their use in mobile devices impractical. Therefore, at least for the foreseeable future, hardware support for low-power microprocessors will play a significant role in the security implementations for Bluetooth solutions. P

## REFERENCES


1. *Specification of the Bluetooth System*, vol. 1.1, 22 Feb. 2001; www.bluetooth.com/dev/specifications.asp.
2. N. Sklavos et al., "Random Number Generator Architecture and VLSI Implementation," *Proc. IEEE Int'l Symp. Circuits & Systems* (ISCAS 02), IEEE Circuits and Systems Soc. Press, Piscataway, N.J., 2002, pp. 854–857.
3. M. Abramovici, M.A. Breuer, and A.D. Friedman, *Digital Systems Testing and Testable Design*, IEEE Press, Piscataway, N.J., 1990.
4. J.L. Massey and R. Rueppel, "Linear Ciphers and Random Sequence Generators with Multiple Clocks," *Advances in Cryptology: Proc. Eurocrypt 84*, Lecture Notes in Computer Science, vol. 209, Springer-Verlag, Berlin, 1984, pp. 74–87.
5. R. Want et al., "Disappearing Hardware," *IEEE Pervasive Computing*, vol. 1, no. 1, Jan.–Mar. 2002, pp. 36–47.
6. P. Chandrakasan, S. Sheng, and R.W. Brodersen, "Low Power CMOS Digital Design," *IEEE J. Solid-State Circuits*, vol. 27, no. 4, Apr. 1992, pp. 473–484.
7. A. Schneier, *Applied Cryptography, Protocols, Algorithms, and Source Code in C*, John Wiley & Sons, New York, 1994.
8. J.L. Massey, G.H. Khachatrian, and M.K. Kuregian, "Nomination of SAFER+ as Candidate Algorithm for the Advance Encryption Standard," *Proc. 1st Advanced Encryption Standard Candidate Conf.*, 1998; www.ee.princeton.edu/~rblee/safer+.
9. J.L. Massey, G.H. Khachatrian, and M.K. Kuregian, "SAFER+ Cylink Corporation's Submission for the Advanced Encryption Standard," *Proc. 1st Advanced Encryption Standard Candidate Conf.*, 1998; http://csrc.nist.gov/CryptoToolkit/aes/round1/conf1/saferpls-slides.pdf.
10. J.L. Massey, "On the Optimality of SAFER+ Diffusion," *Proc. 2nd Advanced Encryption Standard Candidate Conf.* (AES2), 1999; http://csrc.nist.gov/CryptoToolkit/aes/round1/conf2/papers/massey.pdf.