The following paper was originally published in the
Proceedings of the 7th USENIX Security Symposium
San Antonio, Texas, January 26-29, 1998

Secure Videoconferencing

Peter Honeyman, Andy Adamson, Kevin Coffman,
Janani Janakiraman, Rob Jerdonek, and Jim Rees
University of Michigan, Ann Arbor

Secure Videoconferencing†

Peter Honeyman
Andy Adamson
Kevin Coffman
Janani Janakiraman
Rob Jerdonek
Jim Rees

Center for Information Technology Integration
University of Michigan
Ann Arbor

Abstract

At the Center for Information Technology Integration, we are experimenting with algorithms and protocols for building secure applications. In our security testbed, we have modified VIC, an off-the-shelf videoconferencing application, to support GSS, a generic security interface. We then layered these interfaces onto a smartcard-based key distribution algorithm and a fast cipher. Because these components are accompanied by rigorous mathematical proofs of security and are accessed through narrowly-defined interfaces, we have confidence in the strength of the system's security.

1. Introduction

Security and cryptography research and development are advancing at an accelerating rate, yet the payoff in secure distributed applications is not being realized [1, 2]. This failure is due in part to limitations in Internet infrastructure, such as secure naming and routing, which are not to be found except in isolated prototypes.

Progress is being made in securing the essential fabric of the Internet [3, 4, 5], but even these efforts may fail to meet the security needs of the most stringent distributed applications, which must rely on end-to-end methods to satisfy their exceptional security requirements. Security middleware — protocols, algorithms, and interfaces — supports the design and construction of secure distributed applications and provides a context in which the underpinnings of network security can be explored and customized.

† Appears in *Proc.* 17*th* *USENIX Security Conf., San Antonio (January, 1998)*

At the Center for Information Technology Integration, we have built a security testbed that supports prototyping and experimenting with secure distributed applications by providing interfaces to secure communications facilities. The testbed is rich in protocols, ciphers, and other middleware for secure computing; provides for rapid prototyping of secure applications by supporting standards-based interfaces; and allows extensive and flexible performance measurement.

In this paper, we describe a characteristic prototyping activity in the CITI testbed: a secure videoconferencing application. This application has interfaces to the user through a standard graphical user interface, and to peer videoconferencing applications through network communications. We assume good host security, a requirement for secure user interaction, and narrow our attention to threats on networks and remote hosts.

Our approach to building a secure videoconferencing application is to layer its communications needs on secure middleware, using standard interfaces to security functionality (such as encrypting a message or establishing a key) and to communications mechanisms (such as establishing a streaming, half-duplex video interface with a peer). We use a freely available Internet videoconferencing application, to which we add a collection of ciphers and key distribution protocols. The interface between the two is defined with the Internet Generic Security Services standard.

We have good confidence in the security of our videoconferencing prototype. We implement critical cryptographic functionality on secure hardware and access the smartcard with a provably secure key distribution protocol. We hand the session keys that result to a provably secure stream cipher. Cryptographic functionality is accessed through GSS, the sole interface to the cryptographic modules.

The remainder of this paper is organized as follows. In Section 2, we describe the videoconferencing application that we have adapted for secure communications. Section 3 describes the ciphers that meet the performance and security requirements of network videoconferencing. Section 4 details the mechanisms we use for secure key distribution and the role of smartcards. In Section 5, we discuss the interface standard that ties the cryptographic methods to the application. Section 6 describes our experiences with the system and the testbed, and Section 7 discusses future plans.

2. Videoconferencing

With its extreme CPU and I/O performance demands, videoconferencing is a distributed communications service that provides plenty of opportunity for performance measurement and tuning. Consider the data requirements of moving color video images over a data network. Working with full-motion (30 frames per second), 24-bit color, 320×240 images, we need to move almost 7 MBps from the camera, through computers and networks, to the screen. This degree of performance is impossible in most of today's Internet, so we must compress the video stream. Then we need to encrypt it.

We have had good experience with hardware motion JPEG encoders running on IBM workstations, even though these encoders are designed for video capture and storage, not videoconferencing. Nonetheless, we are drawn to the commodity PC market to meet our computing needs. We don't find the fastest computers there, we find the cheapest. Because these computers obey Moore's law, they are increasing exponentially in speed at a constant price.

Cheap MPEG decoders for PCs and laptops are common, thanks to a thriving audio/video playback market. Hardware MPEG encoders remain bulky and expensive while software MPEG encoding demands more cycles than our PCs can provide. We anticipate that the next few years will feature fast, cheap laptops equipped with hardware compression and decompression, but the current state of affairs obeys the maxim ''select two from {good, fast, cheap}.'' So with PCs, encoding is either good or fast; with IBM RS6Ks, we get both, but at considerably higher cost.

As a starting point for our secure videoconferencing prototype, we chose VIC [6], the popular and sophisticated MBONE videoconferencing tool. VIC implements user interface management in Tcl/Tk [7], so it is flexible and easy to extend. VIC offers optional DES encryption of the video stream, but leaves key distribution to users. VIC also includes a weak cipher that XORs the video data stream with a constant key value. This is not secure but provides a best-case performance baseline.

3. Ciphers

DES is not the best choice for encrypting video data. The algorithm is strong enough — it has withstood concentrated attack for over 20 years — but the 56-bit DES key space is fast succumbing to exhaustive search by ever-faster processors. DES is also difficult to implement efficiently in software.

We added two ciphers to VIC: RC4 (see Schneier [8]), a simple stream cipher that is reputed to be fast and secure, and VRA [9], a stream cipher invented by Bellcore cryptographers. VRA is not very widely known, so we describe its operation here.

VRA uses a DES-based Goldreich-Levin [10] pseudorandom number generator (PRNG) as an initial source of random bits. Goldreich-Levin is expensive, requiring one or more calls to DES for each outpit bit, so VRA ''stretches'' these bits into a much longer sequence in two ways. The resulting sequence of pseudo-random bits is then XORed into the data stream.

The first stretching technique uses a long, wide table filled with random bits. A subset of the rows of the table is selected at random and combined with XOR. By selecting in advance the total number of rows n and the number of rows selected at random k, the difficulty of recovering the rows from their XOR sum can be made proportional to a desired $\binom{n}{k}$.

The key to effective stretching is to precompute a wide table, so that a lot of bits are produced from a few calls to Goldreich-Levin. In our application, we use a table with 256 rows, 2,048 bits per row. Initializing this table is expensive, but once built a table can be used without limit in multiple sessions.

This stretching technique exhibits good short-term randomness, with a key strength of approximately $\log(n^k)$, or 48 bits for our choices of n and k, but, like any PRNG, admits a birthday attack [11] that effectively halves the key strength.

To compensate for these long-term correlations, VRA uses a second stretching technique, based on random walks through *expander* graphs. Intuitively, this is a family of sparse graphs with ''dense'' interconnectivity. (A sparse graph is one in which the ratio of edges to vertices is upper bounded by a constant.) By dense interconnectivity we mean that for any division of the vertices into equal-sized subsets, the ratio of the number of edges between them to the number of vertices is lower bounded by a constant.

The essential property of expander graphs is that a short random walk in an expander graph arrives at a truly random node. Specifically, if we start at any of the graph's n vertices and take $\log(n)$ random steps, then the final vertex is very nearly equally likely among all the vertices. A huge value for n foils birthday attacks.

Such a graph is enormous but VRA uses Gabber-Galil expander graphs [12], which can be computed on-the-fly as random steps are made. This ability obviates construction of the entire graph, which is utterly infeasible, and allows the procedure to maintain minimal state, just the neighborhood it is currently traversing. The Gabber-Galil graph we use has $2^{1,024}$ nodes, each node having six neighbors.

To avoid making too many Goldreich-Levin calls, each node on the path of the pseudo-random walk is used as output, producing $\log(n)$ pseudo-random bits at each step. This concession to performance certainly exhibits some short-term correlations, but any outputs more than $\log(n)$ steps apart are essentially independent.

The table and graph techniques produce two streams of pseudo-random bits, one with good short-term characteristics, the other with good long-term ones. These bit streams are XORed together, each masking the others weaknesses. The resulting stream is the ultimate output of the VRA PRNG.

VRA is a keyed PRNG. The key is the set of bits used to initialize Goldreich-Levin, and can be of any size. VRA has essential cryptographic properties, is based on concrete mathematical arguments, and passes numerous tests of randomness, including Knuth's multidimensional tests and Marsaglia's Diehard battery of tests (see [9].) Furthermore, and of utmost importance for our videoconferencing application, VRA is fast.

4. Session keys

Communicating peers establish a security context by agreeing on a shared secret, or *key*, that they use to authenticate and secure subsequent communications. If all principals in a security domain must exchange keys in advance, then the number of keys that must be set up grows quadratically with the number of principals. This does not scale well. The additional requirement that all principals manage a private database of keys makes even small scale deployment uncomfortable.

Needham and Schroeder address these complexities by establishing one long-term key for each of the principals in the security domain and sharing the long-term keys with a trusted third party [13]. This approach has two distinct advantages. First, the number of long-term keys in the system grows linearly with the number of principals, not quadratically. Second, each principal is responsible for only the one key that it shares with the trusted third party, not the keys for all of the other principals in the security domain.

While reliance on a trusted third party reduces the obligations and bookkeeping for principals, it does not eliminate their responsibilities altogether, nor shield them from harm in the event that control over a long-term key is lost. To assist principals in the secure management of their keys, researchers at Bellcore devised an innovative key distribution protocol that exploits the tamper-resistant properties of smartcards to provide a convenient and secure repository for cryptographic keys.

4.1. Smartcards

In the systems we use daily, we find the greatest security threat to be the reliance on passwords selected by users. Users pick passwords that are easy to guess [14, 15]. This is especially troublesome in an environment like ours, which relies heavily on Kerberos IV [16] for basic security services; regrettably, Kerberos IV admits an offline dictionary attack [17]. Smartcards are an attractive technology for reducing or eliminating the reliance on weak, user-selected passwords.

A *smartcard* [18] is a plastic card the size and thickness of an ordinary credit card with electrical contacts and an embedded microprocessor. Putting a computer in everyone's hip pocket creates an infrastructure that enables a huge range of applications, such as vending, personal telecommunications, medical information, home banking and ATM, satellite TV, FAX scrambling, *etc.* The development of smartcard infrastructure provides a context for forward-looking projects such as Xerox PARC's ''Ubiquitous Computing'' initiative. [19].

Smartcards are prevalent in Europe and some other parts of the world, but are an emerging technology in North America. European manufacturers such as SGS-Thomson, Siemens, Gemplus, and Schlumberger are among the most prominent, but Motorola and Texas Instruments are also major players.

The introduction of phonecards over a decade ago paved the way for broad acceptance of smartcards in Europe. European banking and merchant industries have also embraced smartcards, using them in applications such as vending, loyalty card, electronic purses, pay-TV, and identification. By the end of 1997, over a billion smartcards, mostly simple phonecards, were in use worldwide, over 50 million of them advanced, microprocessor-equipped cards [20].

Standardization of smartcard physical characteristics and access protocols [18] plays a vital role in applicability and acceptance. The Europay-MasterCard-Visa and Mondex specifications for smartcard payment systems make it likely that smartcards will continue to play an increasing role in European private and public sectors. The engagement of non-European partners paves the way for global acceptance of smartcards.

A typical smartcard contains an 8-bit microprocessor clocked at 5 Mhz with 8K of EEPROM and a few hundred bytes of RAM, communicating at 9.6 Kbps. Fast DES encryption has long been available [21], and arithmetic co-processors are beginning to be used to provide for subsecond public key operations [22, 23].

Most smartcards have advanced security features to protect the contents of memory from being read or altered by unauthorized users and to protect against improper execution of embedded software. These controls typically include design circuitry to ensure that the embedded software either executes correctly or stops in a safe condition. Critical parameters such as supply voltage, clock frequency, and other critical signals are continuously monitored and filtered to avoid faulty execution [24].

Physical constraints limit the applicability of smartcards in settings that require high-speed computing or communication. But unique security and mobility characteristics make them an attractive foundation for deploying secure distributed applications. For example, smartcards may play the vital role of a trusted computing base in applications that employ downloaded executable content [25], such as Java applets.

4.2. Shoup-Rubin protocol

Before videoconferencers can use VRA, or any other cipher for communications privacy, they need to agree on a session key. Bellcore's Shoup-Rubin protocol [26] is a smartcard-based key distribution protocol that runs among two communicating videoconferencers and a trusted third party. Following Schneier, we call these ALICE, BOB, and TRENT [8].

Shoup-Rubin stores long-term keys on smartcards and performs all cryptography necessary for session key distribution on the smartcards. ALICE never knows her long-term key; it is known only to TRENT and to ALICE's smartcard, where it is used as a key in cryptographic computations.

The role of the Shoup-Rubin protocol is to provide fast and secure session key distribution. The session keys distributed with Shoup-Rubin are not stored on secure hardware, and may be vulnerable to compromise; good practice dictates frequent rekeying.

The details of the Shoup-Rubin protocol are fairly intricate, in part to satisfy the requirements of an underlying complexity-theoretic proof framework. This inconvenience is balanced by the ability to prove powerful properties of the protocol. This mathematical strength, coupled with the hardware basis of long-term key storage, lends good confidence in the overall security of the session key distribution mechanism.

Shoup-Rubin builds on the Leighton-Micali key distribution protocol [27], an inexpensive symmetric key distribution protocol. Leighton-Micali uses a construct known as a *pair key* to establish a shared secret between communicating parties.

Let **A** and **B** denote unambiguous identifiers for ALICE and BOB, and let K_A and K_B be their long term keys, and let $\{M\}_K$ denote message M encrypted with key K. ALICE and BOB's *pair key* is defined

$$\Pi_{AB} = \{\mathbf{A}\}_{K_B} \oplus \{\mathbf{B}\}_{K_A}$$

TRENT calculates pair keys on demand; that is TRENT's entire role. Because a pair key reveals nothing about the long-term keys used in its calculation, it can be communicated in the clear.

With pair key Π_{AB} in hand, ALICE computes $\{\mathbf{B}\}_{K_A}$. Combining this with the pair key yields $\kappa = \{\mathbf{A}\}_{K_B}$. BOB can compute κ directly, so once ALICE has a pair key in hand, she and BOB can communicate privately using key κ.

In Shoup-Rubin, κ is computed on ALICE's and BOB's smartcards. ALICE and BOB then use κ to secure the messages that provide for session key agreement.

The Shoup-Rubin protocol is detailed in the Appendix. Shoup and Rubin use Bellare and Rogaway's innovative complexity theoretic techniques [28] to prove that their key distribution algorithm does not disclose the session key, even to an extremely powerful adversary.

4.3. Shoup-Rubin implementation

The Shoup-Rubin protocol is a distributed computation involving five processing elements: ALICE's computer, her smartcard, BOB's computer, his smartcard, and TRENT. TRENT has access to long-term keys for all the principals in the system.

Working with Personal Cipher Card Corp., a smartcard vendor in Lakeland, FL, CITI implemented the smartcard functionality of Shoup-Rubin on the SGS-Thomson ST16612 card, which contains a MC68HC05 microprocessor clocked at 3.58 Mhz containing 2 KB EEPROM, 6 KB ROM, and 160 bytes RAM. The card supports DES encryption, so that is what we use. Each smartcard call takes about 300 msec. The Shoup-Rubin implementation is about 500 bytes of code, stored in EEPROM.

The total time for key distribution, from the moment a smartcard is inserted into a reader to the time when keys are available, is about 10 seconds. This lengthy delay is due in part to deficiencies in our Windows95 smartcard drivers, but also reflects the message overhead of navigating the ISO 7816 file system on the card. Our goal is one or two seconds on average.

5. Interfaces

To make the encryption and key exchange algorithms available for use in VIC and other applications, we built a Generic Security Service (GSS) [29] interface encompassing the four ciphers (DES, XOR, RC4, and VRA) and Shoup-Rubin. As the name implies, GSS provides security services to callers in a generic fashion, allowing applications to be written to a common portable interface. GSS may be implemented with a range of underlying mechanisms.

The GSS API has four categories of interfaces: credential management, security context, message operations, and support. Shoup-Rubin keeps its credentials on smartcards, so our interface does not implement credential management. A handful of support calls were implemented to handle buffer management and naming issues. Security context establishment and message operations constitute the bulk of our GSS implementation.

We implemented `GSS_Init_sec_context` and `GSS_Accept_sec_context`. The security context interface provides key exchange and establishment of a security context, *i.e.*, the session key, between two entities. The calling applications use the GSS API without knowledge of the underlying mechanisms being used. They call `GSS_Init_sec_context` or `GSS_Accept_sec_context` and pass opaque tokens back and forth until the status values returned indicate that the processing is complete.

We implemented the `GSS_Wrap` and `GSS_Unwrap` message calls. These routines provide for data confidentiality by encrypting the input data. We use the quality of protection parameter, or QoP, to select among encryption methods.

We extended VIC to make GSS calls and augmented its Tcl/Tk interface to allow online cipher selection and performanced data capture. This lets us demonstrate and measure how the choice of ciphers affects the quality of the delivered video.

Implementation of TRENT presents some challenges. On the one hand, TRENT must be online and available at all times. On the other hand, TRENT is entrusted with all of the long-term keys in the system. Combined, these requirements make TRENT a high profile and vulnerable target. We use smartcards to provide for TRENT's seemingly contradictory requirements.

TRENT's function is directory service, so we use an off-the-shelf Lightweight Directory Access Protocol (LDAP) [30] server to provide a standard interface for pair key requests and responses. To minimize the security requirements of the LDAP server, we encrypt the long-term keys with a master key before storing them on the server, and use an attached smartcard for the actual pair key computation. With this approach, the pair-key service can be hosted on a server with security requirements comparable to an email or web server, rather than the extremely stringent security requirements that would be anticipated for a network-attached server holding such vulnerable assets as long-term keys.

6. Testbed Assessment

The CITI security testbed, consisting of a collection of ciphers and key distribution methods tied together with Internet-standard interfaces, supports the development of secure applications. The testbed is easy to extend, and we anticipate adding building blocks.

Our extensions to VIC yield a videoconferencing tool with standard security interfaces, provably secure key distribution, and provably secure end-to-end encryption. Our ability to build, demonstrate, and instrument a prototype implementation validates the usefulness of our security testbed. The tool itself remains very portable and efficient.

Performance measurements were taken from 166 Mhz Pentium systems running Windows '95. The bottlenecks are video encoding and decoding, and Wintel data movement. In these experiments, the presence or absence of encryption makes no difference in throughput. Nonetheless, we are able to measure the time spent in the encryption functions, and thus can estimate the throughput we might expect once we solve our video bottleneck.

Cipher	Throughput
XOR	17 MBps
VRA	3.8 MBps
RC4	1.5 MBps
DES	0.6 MBps

We are encouraged to see VRA outpacing RC4, and are continuing to tune VRA for Wintel.

Our experiences with IBM hardware have been more encouraging. Because we use hardware codecs, we are able to transmit encrypted video at high speed. The following measurements were taken on IBM RS/6000

Model 42T systems running AIX 4.1.2. These are microchannel-based 122 Mhz PowerPC systems with motion JPEG codecs.

Cipher	Throughput	Frames per sec.
XOR	8.9 MBps	30
VRA	1.48 MBps	30
RC4	0.78 MBps	21
DES	0.57 MBps	17

Cipher performance on RS6K is about half that of Pentium because of byte-ordering overhead for the former architecture.

We have used the RS6K-based videoconferencing system to view and control a scanning electron microscope (SEM) in Ann Arbor from a remote location in Washington, DC. Because we had a dedicated 10 Mbps channel, network delay was negligible and jitter nonexistent. Delays otherwise induced by the system were also negligible, and controlling the SEM was a very satisfactory experience. Using VRA encryption, we were able to sustain full-motion video with negligible performance overhead. Using RC4 or DES, however, caused significant degradation of the frame rate.

To highlight the importance of video stream encryption, we built a network snooper that decodes and displays unencrypted video packets. When encryption is enabled, the packet payloads are no longer recognizable as network video, and the image freezes. While unsophisticated, this simple demonstration of the need for secure communications evokes a positive response in its audience.

7. Future work

In our current and future work, we are extending the security boundaries of VIC to include encrypted audio communications. We are also addressing multiparty communications and the attendant problems in secure and reliable group communications. Reliable multicast offers the potential for efficient key distribution to members of a secure session and can play a central role in secure multiparty communications.

Shoup-Rubin needs a mechanism for revocation of long-term keys. If ALICE's long-term key is compromised, BOB may be tricked into establishing a session with an intruder masquerading as ALICE. This comes about because BOB never communicates with TRENT. We are augmenting Shoup-Rubin to preserve security even when smartcards are compromised.

This is an exciting time to be working with secure tokens: new companies and products are making custom programming of secure tokens easy and fast. We are testing Schlumberger's JavaCard and are implementing and augmenting Shoup-Rubin on that platform. Advances such as this pave the way for us to be able to apply the hardware security inherent in secure tokens in a rapid and direct way.

Acknowledgements

S. Rajagopalan, Bill Aiello, and Scott Dexter provided many illuminations. Piet Maclaine Pont of IBM Netherlands suggested the solution to our TRENT dilemma.

Personal Cipher Card Corp. manufactured our Shoup-Rubin smartcards. We thank Kip Wheeler and PC^3 for working so closely with us.

This work was partially supported by a grant from Bellcore.

Availability

Our modifications to VIC are freely available; contact `info@citi.umich.edu`.

References

1. Matt Blaze, ''If cryptography is so great, why isn't it being used more?,'' Invited talk, USENIX Conference, Anaheim (January 8, 1997).
2. Bruce Schneier, *Why cryptography is harder than it looks*, `http://www.counterpane.com/whycrypto.html`, December 23, 1996.
3. John Gilmore, *Secure Wide Area Network Project*, `http://www.cygnus.com/~gnu/swan.html`, March 20, 1997.
4. D. Eastlake and C. Kaufman, ''Domain Name System Security Extensions,'' RFC 2065, USC/ Information Sciences Institute (January 3, 1997).
5. R. Atkinson, ''Security Architecture for the Internet Protocol,'' RFC 1825, USC/ Information Sciences Institute (August 09, 1995).
6. S. McCanne and V. Jacobson, ''VIC: a flexible framework for packet video,'' in *ACM 3rd Ann. Conf. on Multimedia*, San Francisco (November, 1995).
7. J.K. Ousterhout, ''An X11 toolkit based on the Tcl language,'' in *Proc. 1991 Winter USENIX Conf.*, Nashville (January, 1991).
8. B. Schneier, *Applied Cryptography, Second Ed.*, John Wiley & Sons, New York (1996).
9. W. Aiello, S. Rajagopalan, and R. Venkatesan, ''Practical and provable pseudorandom generators,'' pp. 1–8 in *5th Ann. ACM-SIAM Symp. on*

Discrete Algorithms (SODA).

10. O. Goldreich and L.A. Levin, ‘‘Hard core predicates for any one-way function,’’ pp. 25–32 in *21st Ann. ACM Symp. on Theory of Computing* (1989).

11. D.R. Stinson, *Cryptography: theory and practice,* CRC Press, Inc. (1995).

12. Ofer Gabber and Zvi Galil, ‘‘Explicit Constructions of Linear-Sized Superconcentrators,’’ *JCSS* **22**(3), pp. 407–420 (1981).

13. R.M. Needham and M.D. Schroeder, ‘‘Using Encryption for Authentication in Large Networks of Computers,’’ *Communications of the ACM* **21**(12) (December, 1978).

14. Robert T. Morris and Ken Thompson, ‘‘Password Security: A Case History,’’ pp. 594–597 in *CACM* (November, 1979).

15. D.V. Klein, ‘‘Foiling the Cracker: A Survey of, and Improvements to, Password Security,’’ pp. 5–14 in *Proc. UNIX Security Workshop II*, USENIX Assoc., Portland (August, 1990).

16. J.G. Steiner, C. Neuman, and J.I. Schiller, ‘‘Kerberos: An Authentication Service for Open Network Systems,’’ pp. 191–202 in *Winter 1988 USENIX Conference Proceedings*, Dallas (February, 1988).

17. Steven M. Bellovin and Michael Merritt, ‘‘Limitations of the Kerberos Authentication System,’’ pp. 253–267 in *Proc. of Winter USENIX Conf.*, Dallas (January, 1991).

18. International Organization for Standardization, ‘‘Identification Cards — Integrated Circuit(s) Cards with Contacts,’’ ISO 7816.

19. Mark Weiser, ‘‘The Computer for the Twenty-First Century,’’ *Scientific American,* pp. 94–110 (September, 1991).

20. Diogo Teixeira, Cynthia Weaver, and James Beams, ‘‘Smart Cards in Banking: The Future of Money?,’’ 1997 Financial Services Technology Conference, The Tower Group. `http://www.towergroup.com/97conf/97conf.htm`

21. Louis Claude Guillou, Michel Ugon, and Jean-Jacques Quisquater, ‘‘The Smart Card: A Standardized Security Device Dedicated to Public Cryptology,’’ pp. 561–613 in *Contemporary Cryptology: The Science of Information Integrity,* ed. Gustavus J. Simmons, IEEE Press (1992).

22. David Naccache and David M’Raïhi, ‘‘Arithmetic Co-processors for Public Key Cryptography: The State of the Art,’’ pp. 39–58 in *Proc. of CARDIS Smart Card Research and Advanced Applications Conf.,* ed. Pieter H. Hartel, Pierre Pardinas, and Jean-Jacques Quisquater, Stichting Mathematisch Centrum (CWI), Amsterdam (Sept. 1996).

23. Ronald Ferreira, Ralf Malzahn, Peter Marissen, Jean-Jacques Quisquater, and Thomas Wille, ‘‘FAME: A 3rd Generation Coprocessor for Optimising Public Key Cryptosystems in Smart Card Applications,’’ pp. 59–72 in *Proc. of CARDIS Smart Card Research and Advanced Applications Conf.*, ed. Pieter H. Hartel, Pierre Pardinas, and Jean-Jacques Quisquater, Stichting Mathematisch Centrum (CWI), Amsterdam (Sept. 1996).

24. Antony Watts, ‘‘Smartcards and Security — or How to Save $5 Billion a Year!,’’ pp. 102–109 in *Smart Card Technology International*, Chantry Hurst Books, London (1996).

25. TRENT Jaeger, *Flexible Control of Downloaded Executable Content,* PhD Thesis, University of Michigan (1996).

26. V. Shoup and A.D. Rubin, ‘‘Session Key Distribution Using Smart Cards,’’ in *Proc. of Eurocrypt ’96* (May, 1996).

27. T. Leighton and S. Micali, ‘‘Secret-Key Agreement Without Public-Key Cryptography,’’ pp. 456–479 in *Proc. of Crypto ’93,* Santa Barbara (1993).

28. M. Bellare and P. Rogaway, ‘‘Provably Secure Session Key Distribution: The Three Party Case,’’ in *Proc. ACM 27th Ann. Symp. on the Theory of Computing* (1995).

29. J. Linn, ‘‘Generic Security Service Application Program Interface, Version 2,’’ RFC 2078, USC/Information Sciences Institute (January, 10, 1997).

30. W. Yeong, T. Howes, and S. Kille, ‘‘Lightweight Directory Access Protocol,’’ RFC 1777, USC/Information Sciences Institute (March 1995).

Appendix: Shoup-Rubin details

In this section we give a detailed description of the Shoup-Rubin session key distribution protocol. Initially, ALICE and BOB have smartcards initialized with a secret card key and a long-term key shared with TRENT.

The following table defines the terms used in the Shoup-Rubin smartcard-based session key distribution protocol. Encryption of message M with key K is denoted $\{M\}_K$. Integer operands are concatenated to other protocol terms with the ‘‘dot’’ operator to satisfy requirements of the Bellare-Rogaway proof framework.

Term	Meaning
A, **B**	Unique identifiers
K_A, K_B	Long-term keys
K_{AC}, K_{BC}	Secret card keys
r, s	Nonces
$\Pi_{AB} = \{\mathbf{A}\cdot 0\}_{K_B} \oplus \{\mathbf{B}\cdot 1\}_{K_A}$	Pair key
$\alpha = \{\Pi_{AB}\cdot\mathbf{B}\cdot 2\}_{K_A}$	Verifies Π_{AB}
$\beta = \{r\cdot s\cdot 1\}_\kappa$	Verifies r and s
$\gamma = \{r\cdot 1\cdot 1\}_{K_{AC}}$	Verifies r
$\delta = \{s\cdot 0\cdot 1\}_\kappa$	Verifies s
$\kappa = \{\mathbf{A}\cdot 0\}_{K_B}$	See discussion
$\sigma = \{s\cdot 0\cdot 0\}_\kappa$	Session key

The influence of the Leighton-Micali key distribution protocol is evident in the use of ALICE and BOB's *pair key*, defined as

$$\Pi_{AB} = \{\mathbf{A}\}_{K_B} \oplus \{\mathbf{B}\}_{K_A}$$

The pair key allows ALICE and BOB to share a secret without prior agreement.

We now detail the steps of Shoup-Rubin.

From	To	Message	Meaning
ALICE	TRENT	**A**, **B**	ALICE wishes to initiate a session with BOB.
TRENT	ALICE	Π_{AB}, α	Π_{AB} is ALICE and BOB's pair key. α is a verifier for Π_{AB}.

ALICE asks TRENT for the ALICE/BOB pair key. TRENT also returns a verifier, which ALICE's card uses to prevent masquerading.

From	To	Message	Meaning
ALICE	CardA	—	ALICE requests a nonce to verify subsequent communication with BOB.
CardA	ALICE	r, γ	r is a nonce, γ is a verifier for r.

Card operation 1

ALICE initiates the protocol with BOB by requesting a nonce from her smartcard. ALICE retains the verifier for later use.

From	To	Message	Meaning
ALICE	BOB	**A**, r	BOB will use r to assure ALICE of his correct behavior.

By sending a nonce to BOB, ALICE requests establishment of a fresh session key.

From	To	Message	Meaning
BOB	CardB	**A**, r	BOB instructs his smartcard to construct a session key, and provides ALICE's nonce for her subsequent verification.
CardB	BOB	s, σ, β, δ	s is a nonce used to construct the session key. σ is the session key. β is ALICE's verifier for r and s. δ is BOB's verifier for s.

Card operation 2

BOB sends ALICE's identity and her nonce to his smartcard. BOB's card generates a nonce and, from this, a session key. BOB's card also generates two verifiers; one is used by ALICE's card to verify both nonces, the other is used by BOB to verify ALICE's subsequent acknowledgement. BOB retains the session key and his verifier.

From	To	Message	Meaning
BOB	ALICE	s, β	ALICE needs s to construct the session key, and β to verify r and s. BOB retains σ, the session key, and δ, a verifier for s.

BOB forwards his nonce, from which ALICE's card constructs the session key.

From	To	Message	Meaning
ALICE	CardA	**B**, r, s, Π_{AB}, α, β, γ	Verify: Π_{AB} with α, r with γ, and BOB's use of r and s with β. Use Π_{AB} and s to construct the session key.
CardA	ALICE	σ, δ	σ is the session key. δ is sent to BOB to confirm ALICE's verification of s.

Card operation 3

ALICE sends everything she has to her smartcard: BOB's identity, the pair key and its verifier, her nonce and its verifier, and BOB's nonce and its verifier. ALICE's card validates all the verifiers. If everything checks out, ALICE's smartcard constructs the session key from BOB's nonce and uploads it to ALICE along with a verifier to assure BOB that ALICE is behaving properly.

From	To	Message	Meaning
ALICE	BOB	δ	Confirm

ALICE sends the verifier to BOB. BOB compares it to his retained verifier.