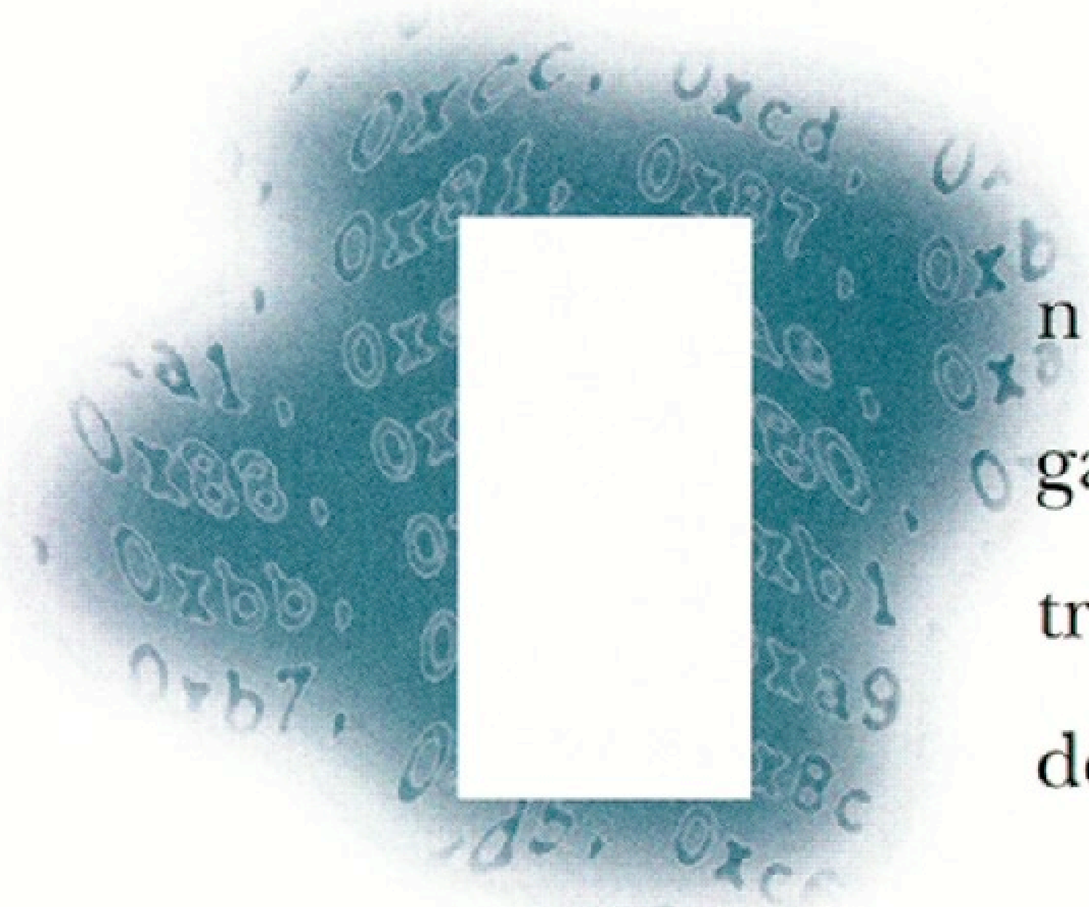
Heralding future reusable security systems, the Yaksha security infrastructure provides multiple security functions—authentication, digital signatures, key exchange, and key escrow.

The Yaksha Security System

Ravi Ganesan



In Hindu methology, Yakshas are "good" demigods who guard the gates of heaven. They are extremely flexible and have the power to transform themselves into other forms, such as birds and tigers. In designing the security infrastructure that will guard the gates to our emerging information infrastructure, it is important that we look for similar flexibility. The Yaksha security system is a security technology [4, 5] capable of reusing a single security infrastructure to perform various security functions—authentication, key exchange, digital signatures, and key escrow. This article describes how the Yaksha security system can be used for key escrow.

It is commonly accepted that encrypted communications and data storage make up an essential component of our emerging information infrastructure. Somewhat more controversial is the concept that certain third parties—other than those communicating or storing information—may have a *legitimate* right to seek access to the information without the active cooperation of the participants. Clearly, encrypting information being communicated or stored could put the third parties at a significant disadvantage. Techniques for providing secure communications and storage with intentional backdoors that allow *legitimate* third parties access to the information fall into the broad category of what may be described as *key escrow* systems. Throughout this article, we use the term *authority* synonymously with *legitimate third party.*

When the authority is the government and the participants are citizens, the entire concept is fairly controversial, as has been well documented in the pages of this magazine [2] and other publications. In this context, the system that dominates the discussion is the so-called Escrow Encryption Standard or Clipper System [3]. An analogous (and in our opinion, less controversial) situation exists when the information is owned by an organization or corporation, the participants are employees, and the third parties are legitimate organizational or corporate authorities. The term "corporate key escrow" is loosely used to describe this situation.

Several key escrow systems have been proposed in recent years, and in this issue of *Communications*, Denning and Branstad [3] present an excellent summary and taxonomy of these various systems. Each of these systems approaches the problem using a different underlying technical approach, as well as from what can be described as a different philosophical stance. The Yaksha system has its own philosophical stance, beginning with a different set of assumptions about the requirements than many of the other systems. The article begins with a summary of key requirements driving the design of the system, provides some necessary background, describes the general system, shows example applications for telephony, email, and data storage, and finally summarizes our conclusions.

Requirements

Two commonly accepted requirements essentially define key escrow systems:

- The system should provide an authority the ability to access encrypted information without the cooperation of the participants.
- The "backdoor" inherent in the system should not be usable by an unauthorized third party.

Although we view the requirements discussed in the following paragraphs as important, they are not necessarily commonly accepted.

Requirement: *Authorities should have access only to short-term session keys, not to long-term user secrets.* In most cryptographic systems, each user has a long-term private secret. In public-key cryptography [10], this would be the user's "private key." In a third-party authentication system [7], this would be the long-term secret shared between the user and the third-party server. For communications security, these long-term private secrets are typically used to negotiate a short-lived session key that in turn is used for encrypting a given session or conversation. If a legitimate authority seeks to eavesdrop on a conversation, one of two things can happen:

- The key escrow system allows the authority to discover the user's long-term private secret, through which the authority can learn the session key for a given conversation and proceed to eavesdrop.
- The key escrow system allows the authority to recover the session key for a particular conversation, but not the long-term private secret. The authority can still eavesdrop successfully, but the long-term private secret is safe.

Key escrow systems should be designed around the latter approach of revealing only short-lived session keys. We believe this is important for several reasons:

- Since the long-term private secret is never revealed to anyone, it can be reused for other functions, such as digital signatures. In systems in which an authority can access this long-term secret, reusing the long-term private secret to generate digital signatures gives the authority the power to forge signatures.
- Revealing only session keys, in our opinion, provides a finer level of granularity of control. For instance, in such systems, one could implement such policies as: The authority can eavesdrop on all of John Doe's conversations, except those he has with his wife or lawyer; or The corporation can decrypt all of John Doe's files saved between March 1994 and September 1994, but not files saved before or after those dates. To our mind, escrow systems represent a compromise between an individual's right to privacy and an authority's right to eavesdrop. Revealing session keys—as opposed to long-term private secrets—provides more opportunities for compromise.
- The compromise is not permanent. That is, in systems in which long-term private secrets are revealed, the compromise of the user's secret is permanent. At some point, the user must get a new private key, or if the key is embedded in a chip in a cellular phone, a new chip. On the other hand, revealing session keys does not compromise the long-term integrity of the permanent secret. So, once the period of "legal eavesdropping" is over, the user does not have to be issued a new private secret.

We note, however, that delivery of session keys to an authority requires the escrow server to be on-line. But we are not suggesting that the escrow agent inspect the contents of any messages; in a practical system, it is unlikely that the agent would have any access to the actual message stream, and the agent's participation would be limited to playing a role in setting up the parameters for a session.

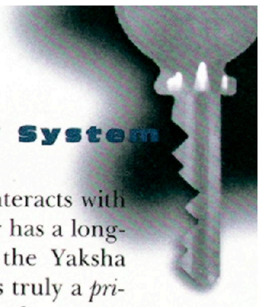
While the justification for this requirement is grounded in a debatable philosophical stance, the next requirement is based on something more concrete—money.

Requirement: *It is very desirable that the key escrow system reuse the security infrastructure necessary for other security functions, such as key exchange, digital signatures, and authentication.*

In theory, it is possible there could be distinct security infrastructures for different security functions. Examples include long-term private secrets to:

- Authenticate yourself (prove your identity) to a bank teller machine;
- Sign a document;
- Perform key exchange for encrypting conversations; and
- Allow you to participate in a key escrow system.

The problem with such an arrangement is cost. Each of these separate keys has an associated infrastructure for generating keys, resetting keys when needed, revoking keys, and so on. From a user perspective, it may well be the case that the distinct infrastructures translate into distinct keys to remember or numerous smartcards to carry. Finally, quite apart from cost, multiple systems increase complexity, which significantly affects the ability to maintain the desired security functionality.



Requirement: *A key escrow system should be implementable in either hardware or software, should apply to both computer communications and telephony, and should be usable both for citizen-government and for employee-organization situations so it has universal applicability.*

This requirement is important for two reasons:

- As we discussed, reusing security infrastructures is beneficial, and it may not be cost-effective to have multiple infrastructures for different types of key escrow.
- There is a clear convergence between telephony and computer communications, and it will become increasingly impractical to treat these situations differently; there is little logic, for instance, in treating voice conversations differently from on-line *chat*.

Is it possible to design a system that meets all these requirements? We believe the answer is yes, and this article describes one system that attempts to meet most of them.

Yaksha

The Yaksha system is based on a variant of the RSA [9] public-key cryptosystem. In public-key cryptography, each user has a long-term private secret key and an associated public key. In the RSA system, the private key of user Alice is a number d_a and her public key is a pair of numbers (e_a,n_a). Similarly, user Bob has a private key d_b and a public key (e_b,n_b). To encrypt a message M, Alice would typically use some encryption function E and a session key k to compute a ciphertext $C = E(M,k)$. To send the message to Bob, she would further encrypt the session key k using Bob's public key and a function known as modular exponentiation, that is $K = k^{e_b} \bmod n_b$. She would then send Bob (C,K). Bob would first recover k using his private key, that is $k = K^{d_b} \bmod n_b$, and can then decrypt the message using a decryption function D, that is $M = D(C,k)$.

In this scenario, Alice and Bob are using a shared session key k for encrypting communications. Recovering M from C without knowledge of k will be very difficult. Such systems, sometimes known as conventional, or single-key, cryptography, are very efficient compared to public-key cryptography. The Data Encryption Standard (DES) [10] is a widely used example. Alice and Bob are, however, using public-key cryptography to exchange the shared session key. In our example, Alice encrypts the session key with Bob's public key and sends it to him. Due to the properties inherent in public-key cryptography, recovering k from K requires knowledge of Bob's private key, which only he knows.

This system achieves privacy using the conventional cryptosystem and key exchange using a public key cryptosystem. How does an authority eavesdrop? One approach would be to split Bob's private key into multiple pieces at key-generation time and escrow it with multiple agencies. Upon getting legal sanction, the authority would collect the pieces, recreate Bob's private key, and then use it to recover the session key k. As discussed earlier, at this point Bob's private key is no longer private. The approach Yaksha uses focuses on revealing the session key k to the authority, thus achieving the authority's objective without compromising the user's long-term private key.

At the heart of our system is an on-line security server, henceforth called the Yaksha server, which interacts with users to perform various functions. Each user has a long-term private secret no one else, including the Yaksha server, knows or can ever reconstruct. This is truly a *private* secret. The Yaksha server maintains for each user (or entity) another long-term private secret. This secret is known *only* to the Yaksha server and is never disclosed to anyone, including the user or authorities. The Yaksha server then interacts with users to perform a number of security functions:

- Providing credentials to authenticate users to other entities;
- Creating joint digital signatures;
- Exchanging session keys in a secure fashion; and
- Acting as a key escrow agent.

This system, which uses a variant of the RSA system, works as follows: as in the RSA system, user Alice has, as her public key the pair (e_a,n_a). Unlike the traditional RSA system, however, the Yaksha system uses two distinct private keys—Alice's private key, denoted by d_{aa}, and the Yaksha server's corresponding key for Alice, denoted by d_{ay}. These two new private keys are related to the original RSA private key d_a by the mathematical relation $d_{aa} \times d_{ay} = d_a \bmod n_a$. For a more complete discussion of this variant and its security properties, please see [1, 4, 5]. This simple, yet powerful, primitive can be used in a variety of complex ways.

In the Yaksha system, each user i has his or her own private key d_{ii}, and the Yaksha server maintains a corresponding d_{iy}. The system can now perform several security functions:

Digital Signatures. Ganesan and Yacobi [5] show how the user can interact with the server to sign a message M. Namely, user Alice performs $S1 = M^{d_{aa}} \bmod n_a$ and sends $S1$ to the Yaksha server. The Yaksha server uses d_{ay} to complete the signature, that is $S = S1^{d_{ay}} \bmod n_a$. Now S is Alice's signature on message M and is indistinguishable from a regular RSA signature. Such a system is of practical importance because by using an on-line server for each signature, we have instant revocation in case a user's private secret is compromised, have a central place to maintain audit trails, and can also (subject to certain restrictions) allow the user's private portion d_{aa} to be a short memorizable password. The last function is critical to implementing digital signatures in an era when smartcards and smartcard readers are not yet ubiquitous. In [5], it is mathematically proven that breaking this system is equivalent to breaking RSA, even in the presence of an active adversary. It is also shown that neither the user nor the server can use knowledge of its private key to determine the private key of the other party; hence the degree of trust in the server is minimized. See [5] for more details.

Authentication. Several authentication protocols are possible using the Yaksha system; we will not describe any in detail here. The interested reader is referred to Ganesan [4], which shows how the Yaksha server can be integrated into the Kerberos [8] third-party authentication system to remove some of the greatest weaknesses of the latter.

Key Exchange. The key exchange system we describe here is chosen to make key escrow possible. When Alice wishes to communicate in private with Bob, she first uses the Yaksha system to negotiate a shared session key. She does this by sending a message to the Yaksha server, expressing her desire to communicate with Bob. The Yaksha server generates a random session key k and computes $C_a = k^{d_{ay} \times e_a} \bmod n_a$ and $C_b = k^{d_{by} \times e_b} \bmod n_b$. It sends C_a to Alice and C_b to Bob. Alice recovers k using her own private key d_{aa}, that is $k = {C_a}^{d_{aa}} \bmod n_a$. Similarly, Bob would recover $k = {C_b}^{d_{bb}} \bmod n_b$. At this point, Alice and Bob have the session key k, and the Yaksha server would destroy its copy of the session key, presumably inside the safe confines of a tamper-resistant chip. For reasons of brevity, we gloss over the fact that in practice k would be encoded as part of a message with a definite structure and a number of other attributes, such as the time stamp. This fact has an important implication: When Alice successfully recovers k from C_a, she has proof that C_a was indeed sent by the Yaksha server.

Key Escrow. At the end of the key exchange protocol, the Yaksha server, which generated the session key, is in a position to provide this key to an authority. This ability forms the basis by which Yaksha can be used as an escrow system. Observe, however, that under no circumstances can the Yaksha server compromise the user's long-term private secret, because the Yaksha server does not *know* this secret. More details are provided in the next section.

In practice, these protocols would be significantly embellished. For instance, it is critical that the session key k be encoded in a data structure with predictable structure. It is also likely that the message structures will contain time stamps. We note, however, that the traditional notion of public-key certificates [6], which provide a mechanism for a user to retrieve another user's public key in a secure fashion, is complementary to Yaksha and would be used as part of the Yaksha system.

Using Yaksha for Key Escrow

The following paragraphs describe how three very different key escrow problems can be solved using the same Yaksha infrastructure. Because our goal is to illustrate the concepts, we do not describe several details, some of which have significant security implications.

Telephony

Our first example is telephony. Since in this model both parties are on-line at the same time, the key exchange protocol described previously can be used exactly as stated. Alice indicates to the Yaksha server a desire for secure communications with Bob. The Yaksha server distributes C_a to Alice and C_b to Bob, who each recover k and use it for encrypting the conversation. In practice, the transaction would be transparent to the user, who might, for instance, simply pick up the phone, dial *007 and then Bob's number, and never notice anything else. The key exchange and other operations would be handled as part of call set-up, and the Yaksha server would be just one more of the many intelligent computers now attached to the phone network to provide special services.

When an authority wishes to tap a phone line, a request R is signed and sent to the Yaksha server. If it is desired that multiple authorities must cooperate to tap a line, we can require that R be signed by multiple authorities. Observe that the authorities are themselves part of the Yaksha system. Each authority has its own private key d_{A1}, d_{A2}, . . . , and the Yaksha server keeps a corresponding d_{A1y}, d_{A2y}, . . . Corresponding public keys (e_{A1},n_{A1}), (e_{A2},n_{A2}), . . . exist in the system. So if, for instance, certain types of taps require the signatures of authorities $A1$ and $A2$, the request sent to the Yaksha server can be of the form $(R^{d_{A1}} \bmod n_{A1})^{d_{A2}} \bmod n_{A2}$. The Yaksha server can authenticate and recover the request using d_{A1y}, d_{A2y}, . . . and the corresponding public keys (e_{A1},n_{A1}), (e_{A2},n_{A2}) . . . The request R can take on several forms, so, for instance, it may order the Yaksha server to provide session keys for all future conversations Alice carries out, or it may ask for only certain types of conversations. The key point to note is that the design provides tremendous flexibility, so a wide variety of underlying policies can be implemented. Further, the policies can be changed easily without huge changes to the system. For instance, if public policy were to change to requiring four cooperating authorities instead of two, or if the identities of the authorities should change, minor changes to the system parameters achieve the goal. The fact that changes can be made easily may not be of great theoretical interest, but as is often observed in practice, it is on such mundane issues as ease of ability to change that the security of systems rests.

It is worth observing that a part of the system requires Bob's telephone to recover k from C_b in a fashion that ensures proof that C_b was generated by the Yaksha server. This observation means it is possible to prevent a dishonest (trying to cheat the key escrow system) Alice from carrying out a secure conversation with an honest (playing by the rules) Bob.

Email

We chose email as our next example because it has a fundamental structural difference from the previous example in the requirements, namely that the underlying messaging is of a store-and-forward nature in which the sender and receiver are not both on-line at the same time. Current systems [6] for secure email are generally based on having the sender, say Alice, sending the receiver, say Bob, the following construct:

$\{\underline{E(M,k)}, \underline{k^{e_b} \bmod n_b}, \underline{S}, \underline{Alice'sCertificate}\}$

The construct has four pieces:

- The message M is encrypted with a session key k generated by Alice.
- The session key k is encrypted with Bob's public key e_b,n_b. On receiving the message, Bob will use his private key d_b to recover this session key k and will then use k to recover M from $E(M,k)$.
- Next, a hash, or fingerprint, $H(M)$, of the message is signed by Alice to generate her signature S, that is $S = (H(M))^{d_a} \bmod n_a$. The hash of the message is used in lieu of the message itself for reasons of efficiency.
- Finally, Alice's certificate (which is simply her public key, in turn signed by a universal authority) is enclosed.

Bob can retrieve Alice's public key from her certificate and use it to verify her signature on the hash.

In keeping with our general policy of integrating Yaksha with existing systems (see [4], where Yaksha is added to Kerberos) as opposed to creating a fresh system from scratch, we attempt to reuse these constructs to the extent possible. We see the system working as follows:

- Alice sends the Yaksha server $S1 = H(M)^{d_{aa}}$ and indicates that the intended recipient is Bob.
- The Yaksha server computes $S = S1^{d_{ay}} \bmod n_a$ and replies to Alice with the message $\underline{S}$, $\underline{k^{d_{ay} \times e_a} \bmod n_a}$, $\underline{k^{d_{by} \times e_b} \bmod n_b}$. The first portion is simply the completed RSA signature for Alice on the message M. The second portion is decrypted by Alice using d_{aa} to recover k. Alice will use this k to encrypt the message M, that is $E(M,k)$. The third portion is sent on to Bob by Alice without modification.
- So the message Alice sends Bob is: $\{\underline{E(M,k)},\underline{k^{d_{by} \times e_b} \bmod n_b},\underline{S},\underline{Alice'sCertificate}\}$. Except for the second field, this is exactly equivalent to the construct Alice would have sent Bob in a non-Yaksha system.
- When Bob receives this message, he verifies Alice's signature exactly as in a non-Yaksha system, but to recover the session key k he uses d_{bb}, that is $k = (\underline{k^{d_{by} \times e_b} \bmod n_b})^{d_{bb}} \bmod n_b$.

Since the Yaksha server has the session key k, the actual escrow process is identical to that described in the telephony example. Some added benefits to this system are that it is now possible to make each user's private long-term secret d_{ii} a fairly short, memorizable password; see [4] for more details. From the standpoint of message structure, the new system is identical to the existing standards [10]. In fact, it is worth observing that interoperability between Yaksha and non-Yaksha systems is relatively easy. If Bob is not a part of the system, his corresponding Yaksha key d_{by} is simply set to one. Bob will not notice the difference, and the escrow will still work. We reiterate that we are glossing over some details essential to secure functioning; for instance, the hash sent by Alice to the Yaksha server should have some specific structure so that the Yaksha server can authenticate Alice before responding.

Encrypted File Storage

As with communications, it is becoming increasingly necessary to provide computer users with access to encrypted files or data storage, and escrow mechanisms are needed. In addition to the usual reasons for an authority to be able to retrieve this data without the user's cooperation, more ordinary reasons, like access to a critical file in a co-worker's absence, also come into play. Using Yaksha to meet this requirement is fairly straightforward; one can think of countless variations. We describe one such possibility, which assumes the existence of a file server process that is an entity independent of the user. The system works as follows:

- Alice sends the Yaksha server the name of the encrypted file server F where she wants to store the file.
- The Yaksha server sends Alice a storage key k encrypted with the Yaksha server's key for the file server, that is $k^{d_{Fy} \times e_F} \bmod n_F$.
- Alice sends the file server the file and the encrypted storage key; note Alice does not know the storage key.
- The file server recovers the storage key using its own private key d_{FF}, encrypts the file with the storage key k, and stores the encrypted file $E(File,k)$ and the encrypted storage key $k^{d_{Fy} \times e_F} \bmod n_F$.
- When Alice wants the file, she simply sends it a signed and time-stamped request Q, which she signs by interacting with the Yaksha server. The file server can verify the signature on the request, recover k from $k^{d_{Fy} \times e_F} \bmod n_F$, decrypt the file, and send it to Alice.
- When an authority wants a file, the authority interacts with the Yaksha server and sends a duly signed request R to the file server. The file server uses the public key(s) of the authority(ies) to verify the signature on the request, recover the session key, decrypt the file, and send it to the authority.

The basic idea is that the file server will only encrypt files using a key it gets from Yaksha. It then stores this key in an encoded form with the file itself. Note that in practice such a system would have provisions for mutual authentication and encrypted communications between the users and the file servers, and most likely would require a signed hash of the file also be stored. All of these functions can be achieved by reusing the Yaksha infrastructure. Observe that we require the file server process to have a long-term private secret key d_{FF}, which it must keep in persistent storage. We anticipate that in a practical system, this key and the functions performed with it will happen inside the safe confines of a tamper-resistant chip. Using a tamper-resistant chip is not particularly onerous, especially since we do not require the storage key for each file to be stored inside the chip. Several variations on this theme are possible.

Conclusions

The Yaksha system requires the presence of an on-line server. In the current climate of cheap and ubiquitous communications, this is (in almost all cases) not a problem. In telephony, for example, on-line servers that provide intelligent services are already ubiquitous. Also consider that today, most credit-card transactions result in access to remote computer systems. Thus, assuming the existence of an on-line service seems to be reasonable; we also assume that the Yaksha server itself will be maintained in a secure fashion. We expect the use of tamper-resistant chips to play a significant role here. For instance, we expect that the Yaksha server's portions of user keys d_{iy} will be encrypted using some sort of master key, which would itself always be stored inside a tamper-resistant chip, and that all the functions performed will happen inside this chip. Given that today's technology allows for systems where every user has a tamper-resistant chip, we do not believe it is too onerous for a few servers to have such chips.

The question then arises: Can colluding cheaters defeat the Yaksha key escrow system? The answer is yes; we do not know of any key escrow system that determined colluding cheaters cannot defeat. The intent here, as in similar systems, is to make it difficult to cheat. We believe this issue is probably addressable at the level of detecting cheating and denying service to cheaters. This is practical in many situations, although an explanation of the techniques is beyond the scope of this article.

The system we describe can be used for many different problem domains, such as secure transmission of movies or software between information providers and set-top boxes in users' homes. The key point in our mind is that the Yaksha system is a single versatile security infrastructure that can be reused for myriad security functions. While not emphasized today, the security infrastructures that will thrive in the future will have the attribute of being reusable.

Finally, we reiterate that flexibility is the single most important factor in a key escrow system. Successful key escrow systems represent a compromise between an individual's rights and an authority's right to know. The Yaksha system provides a flexible alternative that can be adapted to many situations.

Acknowledgment
I would like to thank Dorothy Denning for handling this article for this special section and for her continuing interest in this work.

References

1. Boyd, C. *Digital Multisignatures, Cryptography and Coding.* Clarendon Press, Oxford, 1989. H.J. Beker and F.C. Piper, Eds.
2. Denning, D. To tap or not to tap. *Commun. ACM 36,* 3 (Mar. 1993).
3. Denning, D., and Branstad, D. A taxonomy for key-escrow encryption systems. *Commun. ACM 39,* 3 (Mar. 1996).
4. Ganesan, R. Yaksha: Augmenting Kerberos with public-key cryptography. In *Proceedings of the Internet Society Symposium on Network and Distributed Systems Security,* (Feb.) 1995.
5. Ganesan, R., and Yacobi, Y. A secure joint signature and key exchange system. Bellcore TM-24531, Oct. 1994.
6. Kent, S. Privacy Enhancement for Internet Electronic Mail: Part II: Certificate Based Key Management, Internet RFC 1422, Feb. 1993.
7. Needham, R.M., and Schroeder, M.D. Using encryption for authentication in large networks of computers. *Commun. ACM 21,* 12 (Dec. 1978).
8. Neuman, B.C., and Ts'o, T. Kerberos: An authentication service for computer networks. *IEEE Commun.* (Sept. 1994).
9. Rivest, R., Shamir, A., and Adelman, L. On digital signatures and public-key cryptography. *Commun. ACM 27,* 7 (July 1978).
10. Schneier, B. *Applied Cryptography: Protocols, Algorithms and Source Code in C.* Wiley, New York, 1994.

About the Author:
RAVI GANESAN is Vice President for Information Technology at Bell Atlantic and is with The Johns Hopkins University. **Author's Present Address:** Bell Atlantic, 1320 North Courthouse Road, Arlington, VA 22304; email: ravi.ganesan@bell-atl.com

CALL FOR PAPERS

RE '97

Third IEEE International Symposium on
Requirements Engineering
January 5-8, 1997 • Annapolis, Maryland, USA

The 1997 symposium will be held in four exquisite 18th-century inns clustered in the beautiful colonial seaport of Annapolis on the scenic shores of Chesapeake Bay. It will bring together researchers and practitioners for an exchange of ideas and experiences. The program will consist of invited talks, paper presentations, panels, tutorials, working groups, demonstrations, and a doctoral consortium. The program will also include a parallel *industrial track* with presentations on industry problems and experiences, transferable technology, and commercial tools.

Papers describing original research in requirements engineering are invited. Symposium organizers extend a *special invitation* for paper submission and participation to researchers and practitioners working in *high assurance, safety-critical* and *mission-critical systems,* and *formal approaches* to requirements.

Authors should submit six (6) copies of each full paper (no email or FAX) to the Program Chair. Papers must not exceed 6000 words and must be accompanied by full contact information including name, address, email address, and telephone and FAX numbers. Authors should also submit the title, abstract, and classifications of each paper by email to the Program Chair a month before the paper is due along with full contact information. All papers must be classified according to the symposium classification scheme. For a full call for papers, including the classification scheme, contact the Program Chair, use anonymous FTP from cs.toronto.edu (/dist/ISRE97/CFP), or see the WWW page at http://www.itd.nrl.navy.mil/conf/ISRE97. Developers or researchers wishing to present in the industrial track should submit an abstract to the Industrial Chair. Students interested in presenting at the doctoral consortium should send an extended abstract to the Doctoral Consortium Chair by Sept. 15, 1996.

IMPORTANT DATES:
April 1, 1996: Title, abstract, and classifications due
May 1, 1996: Full papers, industry abstracts due
July 1, 1996: Notification of acceptance
September 1, 1996: Camera-ready copy due

FOR MORE INFORMATION, CONTACT:
Connie Heitmeyer, General Chair
Code 5546, Naval Research Lab, Wash., DC 20375
(202) 767-3596; heitmeyer@itd.nrl.navy.mil
John Mylopoulos, Program Chair
Dept. Computer Sci., Univ. of Toronto, 6 King's College Rd., Rm 283, Toronto, Ontario Canada M5S 3H5
(416) 978-5180; fax (416) 978-1455
jm@cs.toronto.edu
Stuart Faulk, Industrial Chair
Kaman Sciences; (202) 404-6292
faulk@itd.nrl.navy.mil
Myla Archer, Doctoral Consortium Chair
Naval Research Lab; (202) 404-6304
archer@itd.nrl.navy.mil

Sponsored by

IEEE COMPUTER SOCIETY
50 YEARS OF SERVICE •1946-1996
IEEE

IEEE Computer Society TC on Software Engineering
In cooperation with
ACM SIGSOFT, IFIP WG 2.9 (Software Requirements)