Protecting privacy in signal processing

ZHAOHONG WANG AND SEN-CHING S. CHEUNG

© CAN STOCK PHOTO/MAXKABAKOV

While not explicitly stated in the U.S. Constitution, the rights of privacy for many aspects of our lives including religious beliefs, personal possession, and personal information are protected under the Bill of Rights. Nonetheless, news about different forms of privacy invasion has become a daily affair. From the sale of personal information to identity theft, from Google and YouTube surrendering user data to the mining of phone metadata by the National Security Agency, the number of ways that our privacy can be invaded seems to increase at an alarming rate.

One of the reasons for such erosion is the significant advancement in computing technologies for collecting, storing, and sharing personal information among individuals, private sectors, and government agencies. Anyone can now carry thousands of songs, hundreds of pictures, and hours of videos in a small smartphone, ready to be exchanged, sometimes unknowingly, with anyone in the world. The focus of privacy protection often falls on medical or financial records, but it is the multimedia signals—audio, images, and videos—that are driving the entire market of distributed computing while their privacy implications remain poorly understood.

The threats, however, are real. The advance in pattern recognition algorithms such as searchable surveillance or automatic speech recognition systems have turned the once labor-intensive processes into powerful automated systems. They can easily recognize objects of interest like faces, voices, and other biometric signals with high fidelity. Correlating such information with location data such as geo-tags or radio-frequency identification and other information on social networks allow hackers to easily track the activities and associations of any individuals. The matter is further complicated by the unprecedented effort of the government in monitoring the activities of private citizens to fight terrorism. It is thus imperative to develop a comprehensive privacy protection framework for personal multimedia data without jeopardizing our homeland security.

While new legislature and policy changes are essential elements of such a framework, technologies are playing an equally pivotal role in safeguarding private information. There are two main technical challenges in protecting multimedia data. First, as signal capturing devices and wireless networks become ubiquitous, diverse applications from multimedia e-mails and blogging to large-scale surveillance networks begin to demand some form of privacy protection. A comprehensive framework is needed to identify appropriate sensitive information in different applications and to provide different levels of protection depending on the role of each user in the system.

Second, from simple enhancement to sophisticated pattern recognition, multimedia data requires various signal processing operations to become useful. The current cloud and peer-to-peer computing platforms have provided ubiquitous data storage for multimedia data. In the future, software developers will undoubtedly take advantage of the enormous power of various distributed computing platforms in offering different types of software services to process these data. It is easy to see why privacy is needed in any distributed multimedia processing service platform—a user may want to enhance or process a video that was taken using a smartphone but lacks the

Digital Object Identifier 10.1109/MPOT.2013.2295652
Date of publication: 30 April 2014

required capability and algorithms. A multimedia software vendor offers proprietary software for the job and charges the user based on the duration of the video needed to be processed. The two parties come to the cloud server and take advantage of its enormous power and storage. The user clearly wants to protect any private information in his/her video while the software vendor needs to prevent the theft of its proprietary algorithm. The key challenge is to design an appropriate privacy protection scheme so that it will not compromise any legitimate processing of the data.

In this article, we provide a short tutorial on secure multiparty computation (SMC) for providing privacy protection of sensitive information in distributed processing of multimedia signals. SMC is an active area of research in cryptology. In recent years, it has been successfully applied to solve many privacy protection problems in distributed signal processing including face recognition, iris matching, image denoising, video surveillance, visualization, and information retrieval. Our goal is to introduce the core building blocks of SMC that can be used to build sophisticated signal processing algorithms.

From the sale of personal information to identity theft, from Google and YouTube surrendering user data to the mining of phone metadata by the National Security Agency, the number of ways that our privacy can be invaded seems to increase at an alarming rate.

Background

Following the example introduced in the previous section, we consider a distributed computing task to have at least two participants: 1) a user with a private input requires a service from 2) a vendor with a proprietary software algorithm. For simplicity, we assume that the vendor's secrets are the key parameters used in an otherwise well-known algorithm. For example, the parameters could be the tap values of a sophisticated filter or thresholds and weights in a neural network. The privacy objective is as follows: the user and the vendor do not trust each other and would like to prevent each other from knowing anything about their own secret information.

The simplest form of an SMC is a two-party SMC, as shown in Fig. 1(a), where the information exchanged between the two parties does not disclose any information about the secrets. This is the model used in encrypted-domain techniques, with the most commonly used one being the homomorphic encryption (HE) described in the section "Homomorphic encryption." An HE system allows operations to be performed directly on encrypted signals.

In mobile applications, either the user or the vendor may have enough computation power to carry out the task. As such, some of the computation needs to be outsourced to a centralized cloud computer or a distributed peer-to-peer network of computers. This is a three-party SMC, as shown in Fig. 1(b). HE can be used for the three-party case as well but a more efficient technique called Shamir's secret sharing (SSS) can be used. SSS will be introduced in the section "Shamir's secret sharing scheme." While the three-party SMC is more flexible in offloading complex computation, it involves more participants in the computation which pose additional requirements in securing the communication.

To clearly understand how these techniques can provide security to private data, we have to clarify what "security" means. It is derived from Shannon's secure communication model as shown in Fig. 2. This model consists of a sender A, wanting to send a random message m to a receiver B, through an insecure channel. The message m is called a plaintext taken from a finite set, called plaintext space $\boldsymbol{M}$.

Since the channel is insecure, the transmission between A and B may be intercepted by an eavesdropper E. To protect the message, A applies a mapping E_k on m to generate c, i.e., $E_k(m) = c$. E_k is called an encryption function. k is called the key, also treated as a random quantity taken from the key-space $\boldsymbol{K}$. c is the ciphertext, which is an element of the ciphertext space $\boldsymbol{C}$. B must be able to retrieve the plaintext message m from the ciphertext c by means of the decryption function D_k: $D_k(c) = m$. This is a symmetric cryptosystem in which A and B share the same secret key which is not available to E. The cryptosystem system is called *information-theoretic secure* if the ciphertext c is statistically independent of the plaintext m. For an information-theoretic secure cryptosystem, it is impossible for the adversary E to learn any information about m from c, regardless of the capability of E.

In practice, symmetric cryptosystems are not very useful for secret communication. First, a separate secure channel must exist for the exchange of secret keys. Second, the key must be randomly selected for each message so as to maintain statistical independence. Instead, asymmetric or public-key cryptosystems are used, where the encrypting key is different from the decrypting key. The encrypting key is made public so that any sender can send secret message to the receiver. Unfortunately, there is no known public-key cryptosystem that is information-theoretic secure. The best schemes can only provide computational security—the adversary cannot decrypt the ciphertext provided that she only has access to "limited" computational resource. The security of computational-secure cryptosystems strongly depends on the length of the key and it is not uncommon for practical systems to use keys with hundreds or even thousands of bits. Factoring in the progress of central processing unit speeds, the security of a computational-secure cryptosystem is usually specified by an estimate of the year when it can be hacked by the most powerful computer. For example, it is estimated that the 2048-b RSA system will become insecure by the year 2030.

Another dimension of security is the modeling of adversarial actions. If all E does is to eavesdrop without disturbing the flow of information, such adversarial behaviors are described as semihonest. On the other hand, a malicious adversary may inject wrong information or hack into the sender/receiver to terminate the transmission or computation prematurely. Such malicious behaviors are difficult to combat. We will only focus on the semihonest adversaries.

The SMC paradigm extends the aforementioned secure communication model to include computation. SMC specifies the computation and communication procedure for each of the participants involved. Like the communication model, all information exchanged among parties is encrypted to protect against eavesdropping. Unlike the communication model, the goal of SMC is not to communicate but to process the input signal and produce an output signal for the user. As such, the receivers are treated as adversaries and do not possess the key to decrypt the ciphertext. Instead, the specially designed cryptosystems used in SMC possess a special property called homomorphism that allows the receivers to operate directly on the ciphertext.

For example, the user wants the receivers to compute a function $f(x_1, x_2)$ on the two plaintext numbers x_1 and x_2 but the receivers only have access to the corresponding ciphertext $c_1 = E(x_1)$ and $c_2 = E(x_2)$. A homomorphic cryptosystem supports a ciphertext function $g(c_1, c_2)$ such that

$$g(c_1, c_2) = E(f(x_1, x_2)).$$

By executing $g(c_1, c_2)$ and then sending the result back to the user for decryption, the receivers achieve the computation goal without learning anything about the inputs.

Next, we discuss how homomorphism is realized in both HE and SSS. For simplicity, we only consider how addition and multiplication on fixed-point numbers can be realized in an encrypted domain. More sophisticated algorithms, however, can be realized by using these operations as building blocks.

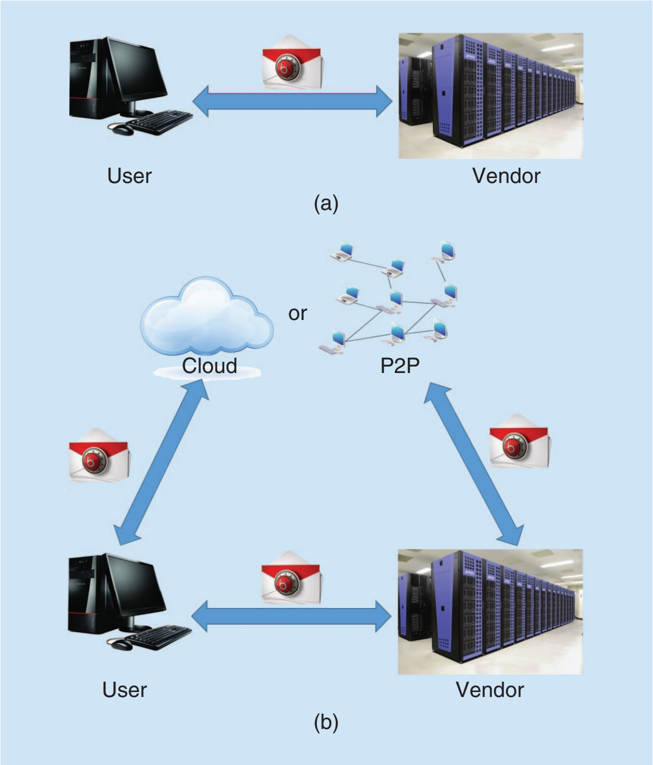

Fig. 1 (a) Two-party and (b) three-party SMCs.

Homomorphic encryption

HE is a family of public-key cryptosystems that can be used in two-party SMC. The security is computational, and its strength depends on the key length and the hardness of the underlying mathematical construction. Until recently, most HE systems were homomorphic with respect to either addition or multiplication but not both. The first result of a fully HE (FHE) was demonstrated by Craig Gentry in 2009. This discovery has sparked a flurry of research activities but a practical FHE implementation remains elusive. As such, we will focus on a popular HE called the Paillier system, which is homomorphic with respect to addition. We will describe the mathematical principles behind Paillier, its additive homomorphism, and how it can be extended to handle multiplication.

Like many other cryptosystems, Paillier is defined using modular arithmetic. In modular-N arithmetic, any integer a is equivalent to its remainder b in $\{0, \ldots, N-1\}$ after dividing by N, i.e., $a = kN + b$ for some integer k. We use the following notation to denote this equivalence relation:

$$a \equiv b \bmod N.$$

A special function $\varphi(N)$ in modular arithmetic, called the Euler's totient function, plays a very important role in the Paillier system. It represents the number of positive integers smaller than N that are relatively prime to N. One of its key properties, stated here without proof, is that

$$r^{\varphi(N)} \equiv 1 \bmod N \tag{1}$$

for any positive integer r relative prime to N. When N is a product of two very large primes (A secure Paillier system requires the key to be between 1,024 and 2,048 b, resulting in N between 10^{308} and 10^{617}), computing its Euler's totient function is very difficult. It is the difficulty of this problem that provides the computational security of the Paillier system.

In the Paillier system, the public key for encryption is N, and the private key for decryption is $\varphi(N)$. Given a plaintext number x in $\{0, \ldots, N-1\}$, the encryption is defined as follows:

$$E(x, r) \equiv (1+N)^x \cdot r^N \bmod N^2,$$

where r, relatively prime with N, is randomly selected for each invocation of the encryption function. The encryption formula is the key to understanding Paillier. First, notice that the modulus changes from N to N^2. This allows for easy decryption as we will later explain. Second, the secret number x is hidden in the power of the constant $1+N$. This enables additive homomorphism because the ciphertext of the sum of two plaintext numbers is precisely the product of the two corresponding ciphertext:

$$\begin{aligned} E(x_1, r_1) \cdot E(x_2, r_2) &\equiv (1+N)^{x_1} \cdot r_1^N \\ &\quad \cdot (1+N)^{x_2} \cdot r_2^N \bmod N^2 \\ &\equiv (1+N)^{x_1+x_2} \cdot (r_1 r_2)^N \\ &= E(x_1 + x_2, r_1 r_2). \end{aligned}$$

Third, the use of the random number r in the encryption process is particularly crucial. This number is selected by the sender and is not shared with anyone. It essentially randomizes the ciphertext to an extent that the ciphertext for the same plaintext x would be different every time the encryption function is called. This prevents the so-called chosen plaintext attack in which the eavesdropper attempts to hack the system by building a lookup table between known plaintext and the corresponding ciphertext.

On the other hand, the randomness injected to the plaintext must be removed during the decryption process. This is where the private key $\varphi(N)$ comes in. The first step of the decryption is to raise the ciphertext by a power of $\varphi(N)$:

$$E(x, r)^{\varphi(N)} \equiv (1+N)^{x\varphi(N)} \bullet r^{N\varphi(N)} \bmod N^2.$$

There are two terms in the product on the right-hand side of the equation. Let us consider the second term $r^{N\varphi(N)}$ first. From (1), we know that $r^{\varphi(N)} = 1 + kN$ for some positive integer k. Thus,

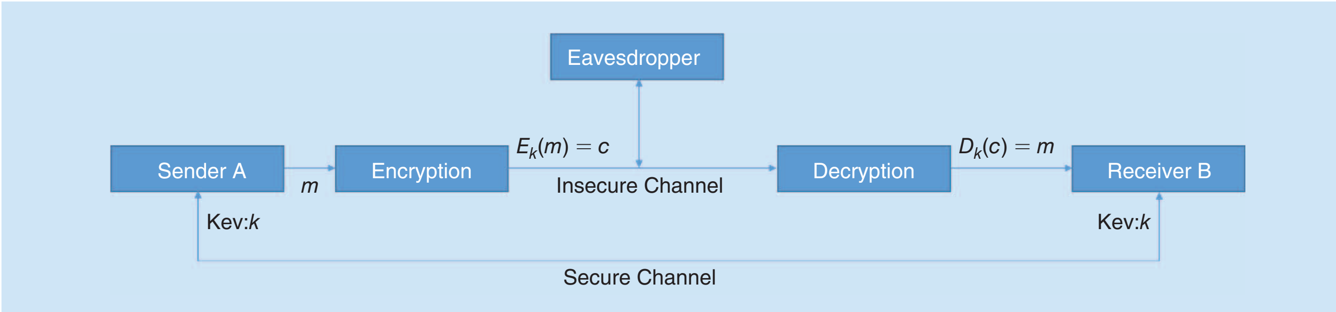

Fig. 2 Shannon's model for defining communication security.

$$\begin{aligned} r^{N\varphi(N)} &= (1+kN)^N \\ &= \begin{array}{l} 1+kN^2+\text{multiples of } N^k \\ \text{with } k > 2 \end{array} \\ &\equiv 1 \bmod N^2. \end{aligned}$$

The second equation is based on the binomial expansion. As all but the first term have N^2 as a factor, taking modulo-N^2 eliminates them, reducing the result to 1. In other words, raising the ciphertext by the power of the private key annihilates the randomness injected into the ciphertext. Coming back to the first term $(1+N)^{x\varphi(N)}$, we can again use the binomial expansion to simplify

$$\begin{aligned} (1+N)^{x\varphi(N)} &= 1 + xN\varphi(N) + \text{multiples} \\ &\quad \text{of } N^k \text{with k} \geq 2 \\ &\equiv 1 + xN\varphi(N) \bmod N^2. \end{aligned}$$

Combining the simplification of these two terms together, we obtain

$$E(x,r)^{\varphi(N)} \equiv 1 + xN\varphi(N) \bmod N^2.$$

Now, the decryption of $E(x,r)$ can be easily realized as follows:

$$x \equiv \frac{E(x,r)^{\varphi(N)} - 1 \bmod N^2}{N\varphi(N)} \bmod N^2.$$

The fraction on the right-hand side is computed in the integer domain as N does not have a multiplicative inverse in modular-N^2 arithmetic. This completes the decryption process.

The additive homomorphic property is very useful in performing joint computation. As mentioned before, given two ciphertext numbers (to simplify our notation, we have dropped the random component in the encryption as it does not affect the result) $E(x_1)$ and $E(x_2)$, the receiver can implement plaintext addition as $E(x_1)\cdot E(x_2) = E(x_1 + x_1)$. If the receiver also has a private input a itself, it can be added to x_1 by first encrypting it as $E(a)$ and then computing $E(x_1)\cdot E(a) = E(x_1 + a)$. The receiver can also compute ax_1 in the encrypted domain by computing $E(x_1)^a = E(ax_1)$. The only problem is that there is no straightforward operation to directly compute the product $x_1\,x_2$ in the encrypted domain.

The only solution is to send the encrypted data back to the sender who possesses the private key for decryption. However, $E(x_1)$ and $E(x_2)$ cannot be sent back directly to the sender as they might have already accumulated private inputs from the receiver. Instead, the receiver needs to first perturb them by adding random noise s_1 and s_2 to create $E(x_1+s_1)$ and $E(x_2+s_2)$ respectively before sending them back to the sender. The sender decrypts them, computes the product $(x_1+s_1)(x_2+s_2)$, encrypts the result, and then sends it back to the receiver. In the final step, the receiver computes $E(x_1\,x_2)$ using

$$\begin{aligned} E(x_1x_2) &= E[(x_1+s_1)(x_2+s_2) \\ &\qquad -x_1s_2 - x_2s_1 - s_1s_2] \\ &= E[(x_1+s_1)(x_2+s_2)] \\ &\qquad \cdot E(x_1s_2)^{-1}\cdot E(x_2s_1)^{-1}\cdot E(s_1s_2)^{-1}. \end{aligned}$$

In the last expression, the first term of the product comes from the sender while the remaining three terms are all locally computed in the encrypted domain by the receiver. This somewhat complicated approach of realizing multiplication in the encrypted domain is one of the reasons behind the high computational cost of using Paillier system for two-party SMC.

Shamir's secret sharing scheme

To ensure computational security, the modulus N^2 used in the Paillier system can go as high as $2^{4,096}$. In other words, we will need 4,096 b to hold just a single number for computation. This is significantly higher than the 32- or 64-b architecture supported by most of the current microprocessors. Signal processing algorithms often require real-time processing of a large amount of data and the high computation cost of Paillier is thus a significant hurdle for practical applications.

If there exists an additional participant who promises not to collude with either the user or the vendor in stealing the other's secret, there exists a more efficient alternative: SSS. Secret sharing is different from encryption/decryption in that it breaks a secret number into many numerical shares. Any party can reconstruct the secret if it has enough number of shares but otherwise knows nothing about the secret.

In SSS, the secret number x in $\{0,\ldots,N-1\}$ is hidden as a constant term of a random $(t-1)$- degree polynomial

$$\begin{aligned} g(s) \equiv r_{t-1}s^{t-1} + r_{t-2}s^{t-2} \\ + \cdots + r_1 s + x \bmod N, \end{aligned}$$

where r_i for $i = 1,\ldots,t-1$ are random numbers chosen by the secret owner. Such a polynomial is fully specified by its evaluations at different points: $g(0)$, $g(1),\ldots,g(N-1)$. Besides $g(0) = x$, the rest of these evaluations are called secret shares of x. If N is a prime number, x can be reconstructed using any collection of $k \geq t$ shares $g(s_1), g(s_2),\ldots, g(s_k)$ based on the following formula:

$$x \equiv \gamma_1 g(s_1) + \cdots + \gamma_k g(s_k) \bmod N, \quad (2)$$

where

$$\gamma_i \equiv \begin{pmatrix} (-s_1)\ldots \\ (-s_{i-1}) \\ (-s_{i+1})\ldots \\ (-s_k) \end{pmatrix} \Bigg/ \begin{pmatrix} (s_i - s_1)\ldots \\ (s_i - s_{i-1}) \\ (s_i - s_{i+1})\ldots \\ (s_i - s_k) \end{pmatrix} \bmod N$$

for $i = 1,\ldots,k$. This is called the Lagrangian interpolation formula.

On the other hand, any collection of less than t secret shares provide no information about x whatsoever, i.e., for any secret number, one can find a random polynomial with that secret number as the constant term that is compatible with the given collection of shares. Such lack of knowledge about the secret is a mathematical fact and has nothing to do with the computational capability of the adversary. As such, SSS is information-theoretic secure. Now let us see how SSS can be used in a three-party SMC.

Suppose the user U has a secret number x and the vendor V has a secret number y. For a three-party SMC, there is also a third participant C involved in the computation who promises not to collude with either U or V. To create secret shares, U randomly generates a first-order polynomial $f(s) = r_1 s + x$ and computes secret shares $f(1), f(2)$, and $f(3)$. U keeps $f(1)$, and sends $f(2)$ to V, $f(3)$ to C. As long as V and C are not colluding, they can learn nothing about x. Similarly, V randomly generates a first-order polynomial $g(s) = r_2 s + y$ and computes secret shares $g(1), g(2)$, and $g(3)$. To allow computation on the secret shares, it is important that the secret shares held by a particular participant are all evaluated at the same point. Thus, V will send $g(1)$ to U and $g(3)$ to C. The additive homomorphism in secret shares can be easily realized by having each participant add up their shares, i.e., U computes $f(1) + g(1)$, V computes $f(2) + g(2)$, and C computes $f(3) + g(3)$. These are the three secret shares of the function

$$f(s) + g(s) \equiv (r_1 + r_2)s + x + y \bmod N, \tag{3}$$

which hides the desired $x + y$ as the constant term.

SSS is multiplicative homomorphic as well. Each participant can multiply their own shares, which become $f(1)\,g(1)$, $f(2)\,g(2)$, and $f(3)g(3)$. They are the secret shares of the function

$$f(s)g(s) \equiv (r_1 r_2)s^2 + (r_2 x + r_1 y)s + xy \bmod N, \tag{4}$$

which hides the desired xy as the constant term. The key difference between SSS's additive and multiplicative homomorphism is the degree of the resulting polynomial. The degree of the additive polynomial in (3) is still one while the degree of the multiplicative polynomial in (4) becomes two. This has an important consequence: U can always reconstruct the final answer regardless of the number of additions by collecting one additional share from either V or C. However, only one multiplication can be performed as further multiplication will increase the degree beyond two, rendering the final result unreconstructible with only three secret shares.

To support more than one round of multiplication, we need to reduce the degree of the multiplicative polynomial using a two-step process called renormalization. First, each participant treats the share product $f(i)g(i)$ as a secret and generates three secret shares for that. Specifically, U creates $h_U(s) = r_U s + f(1)g(1)$ and generates secret shares $h_U(2)$ and $h_U(3)$ for V and C respectively. Similarly, V creates $h_V(s) = r_V s + f(2)g(2)$ and generates secret shares $h_V(1)$ and $h_V(3)$ for U and C, and C creates $h_C(s) = r_C s + f(3)g(3)$ and generates secret shares $h_C(1)$ and $h_C(2)$ for U and V. No secret information is exchanged as none of the participant receives more than one share from the same function.

Second, U, V, and C compute the final secret shares as

$$h(i) := \gamma_1 h_U(i) + \gamma_2 h_V(i) + \gamma_3 h_C(i) \bmod N$$

for $i = 1,2,3$. γ_is are the coefficients used in the reconstruction formula in (2). These secret shares correspond to a function $h(s)$, which can be derived in the boxed equation at the bottom of the page. The second line is based on the definitions of $h_U(s), h_V(s)$, and $h_C(s)$. The third line is based on the reconstruction formula in (2). The renormalization procedure produces a polynomial with degree one and the product of the two secret numbers as the constant term. As such, more multiplication operations can be applied.

$$\boxed{\begin{aligned} h(s) &\equiv \gamma_1 h_U(s) + \gamma_2 h_V(s) + \gamma_3 h_C(s) \bmod N \\ &\equiv (\gamma_1 r_U + \gamma_2 r_V + \gamma_3 r_C)s + (\gamma_1 f(1)g(1) + \gamma_2 f(2)g(2) + \gamma_3 f(3)g(3)) \bmod N \\ &\equiv (\gamma_1 r_U + \gamma_2 r_V + \gamma_3 r_C)s + xy \bmod N. \end{aligned}}$$

Compared with HE, SSS is significantly faster because the calculations can be done directly on the signal samples. Rather than using operands with thousands of bits to provide computational security, the computations of SSS can all be done using, say 16-b numbers if the signal samples are of 8-b precision. The extra 8 b can be used to prevent possible overflow during the computation process.

Collusion deterrance

The security of the three-party SSS example hinges on the assumption that C does not collude with either U or V. In practice, it is very difficult to enforce this assumption –C may be colluding with U to steal V's secret using a communication channel different from that used in the SMC. The communication messages and the results within the SMC remain completely unchanged and as such, the act of collusion is completely undetectable. This is the Achilles' heel of SSS and a number of approaches have been proposed to mitigate this problem. In this article, we briefly describe our approach on using a game theoretic design in deterring collusion behaviors.

The key to our design is that while colluding to steal another's secret may have a good payoff, there needs to be a venue for the victim to "retaliate." Let us assume that all participants are rational—they choose among the two possible strategies, honest or cheating, to optimize their own payoff. A possible venue that supports SSS-based computation can go as follows. As the key objective for any SMC is to protect privacy, U and V must first agree to abide by a legal-binding contract to pay for damages if caught colluding with C. Suppose that after executing the computation, V accuses U of trying to steal his secret. U is likely to defend himself with different tactics and may even accuse V back for stealing his secret. A judgment will ultimately be rendered by an appropriate authority after possibly a long proceeding to evaluate all the available evidence. The extra cost and effort of collecting evidence and going through the proceeding makes retaliation the most undesirable outcome for everyone.

This disastrous outcome is not the only scenario. To fully understand the tradeoffs between choosing the two strategies, each participant must rank all possible outcomes so as to choose the strategy that results in the best payoff. A reasonable ranking of different outcomes is shown in Table 1. As postulated before, the lowest ranked outcome is retaliation brought on by the cheating strategy of either U or V. The second lowest-ranked outcome is when both

Table 1. The ranking of different outcomes.

Strategies	Retaliate?	*U* Rank	*V* Rank
Either or both cheats	Y	1	1
Both cheat	N	2	2
U cheats only	N	5	3
V cheats only	N	3	5
No one cheats	N	4	4

Table 2. The pay-off matrix.

		Vendor V	
		Honest	**Cheating**
User *U*	**Honest**	(p_4, p_4)	$(1-q)(p_1, p_1) + q(p_3, p_5) = (qp_3, q)$
	Cheating	$(1-q)(p_1, p_1) + q(p_5, p_3) = (q, qp_3)$	$(1-q^2)(p_1, p_1) + q^2(p_2, p_2) = (q^2 p_2, q^2 p_2)$

have cheated but neither retaliates—even though both U and V steal each other's secrets without getting caught, the fact that they both cheat would imply that they have wasted resources colluding with C in stealing something that is of little value. If only U cheats and gets away with it, U will have the highest-ranked outcome. As for V, we give a rank of 3 for two reasons: 1) V successfully carries out the task and gets compensated and 2) V does not retaliate because either a) he is unaware of the theft as he does not put in a significant effort in tracking any leakage of his secret or b) the cost of retaliation is higher than the cost of his secret. Either reason implies that the loss of the secret may not be too significant to V. The situation is reverse if we switch U and V. Finally, we assign the second highest rank of 3 to both U and V when they complete the task faithfully.

While we believe that these rankings are general for most applications, mapping them to numerical payoff values depends on the values of the secrets and the computational task. For simplicity, we assume that the same payoffs are used for both U and V and denote the payoff values as $0 = p_1 < p_2 < p_3 < p_4 < p_5 = 1$ for the five different ranks. Notice that we normalize all the payoff values within 0 and 1 as only the relative values matter in determining the optimal strategies.

Since the outcome depends on the decision to retaliate, which in turn depends on how a player values his/her secret compared to the cost of retaliation, we define a "nonretaliate" probability q for both U and V, conditioned on the discovery of others' cheating behavior. For example, $q = 1$ means that no one retaliates or cares about the secret, while $q = 0$ means that a player will always retaliate if his/her secret is stolen. For a typical SMC application, q should be close to 0 as all secrets are highly valued and any leakage of secret is easily identifiable.

Using a standard representation in game theory, we compute the payoff matrix for this game in Table 2. The two-tuple in each entry indicates the average payoffs of U and V when adopting the row and column strategies respectively. For example, the lower left outcome corresponds to the case when U is cheating and V is honest. With probability $(1-q)$, V will retaliate resulting in the worst payoff (p_1, p_1). With probability q, V does not retaliate and U gets away with the theft, resulting in the payoff of p_5 for U and p_3 for V.

With this payoff matrix, we can compute the celebrated Nash equilibrium (NE) for this game. The significance of the NE is that it represents the best strategy taken by a participant regardless of the choice of the other. Interested readers should refer to any game theory books on how to find the NE. For this game, one can show that the NE would be for both U and V to stay honest, provided that $p_3 \geq q$. As we have argued that q is small for SMC applications, this result indicates that both participants will stay honest given the threat of retaliation. The situation becomes more complicated if $p_3 < q$ and is beyond the scope of this article.

Conclusions

We have motivated the importance of privacy protection in processing multimedia signals in distributed networks. The computational framework to enable such protection is secure multiparty computation. For two-party SMC, HE is often used and we have discussed the mathematics behind one of the most popular HE called the Paillier cryptosystem. If more noncolluding participants are available, Shamir's secret sharing can deliver better performance and can guarantee information-theoretic security. To deter possible collusion attacks, we have presented a game-theoretic construction and showed that staying honest is the NE. While our discussions remain at a rather theoretical level, exciting practical applications have begun to emerge in the past few years. It is our hope that these technologies can make our world safer without sacrificing our precious right to privacy.

Acknowledgment

Part of this material is based upon work supported by the National Science Foundation under grant number 1018241. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

Read more about it

The March 2013 issue (vol. 30, no. 2) of *IEEE Signal Processing Magazine* is dedicated to the latest research in the field of signal processing in the encrypted domain and contains many articles relevant to this article.

For an excellent introduction of the fundamental concepts in cryptography including a discussion on Paillier cryptosystem, we recommend

- J. Katz and Y. Lindell, *Introduction to Modern Cryptography: Principles and Protocols*. London, U.K.: Chapman and Hall, 2007.

For an introduction of the secure multiparty computation and SSS, we recommend

- R. Cramer and I. Damgard, "Introduction to secure multi-party computations," in *Contemporary Cryptology* (Advanced Courses in Mathematics CRM Barcelona). Switzerland: Birkhauser, 2005, pp. 41–87.

For a classical introduction to game theory, we recommend

- J. N. Webb, *Game Theory: Decisions, Interaction and Evolution*. London: Springer-Verlag, 2006.

About the authors

Zhaohong Wang (zhaohng.wang@uky.edu) is currently a Ph.D. candidate in electrical engineering at the University of Kentucky. His research interests include cybersecurity, secure computation, and cloud computing.

Sen-ching S. Cheung (sccheung@ieee.org) is an associate professor in the Department of Electrical Engineering at the University of Kentucky. He earned his Ph.D. from the University of California, Berkeley, in 2002.