# DECOY-BASED INFORMATION SECURITY

VLADIMIR SHPILRAIN

Where does a wise man hide a leaf? In the forest.
But what does he do if there is no forest?
He grows a forest to hide it in.
G. K. Chesterton, *The Innocence of Father Brown*

Abstract. In this survey, we discuss an emerging concept of decoy-based information security, or security without computational assumptions. In particular, we show how this concept can be implemented to provide security against (passive) computationally unbounded adversary in some public-key encryption protocols. In the world of symmetric cryptography, decoy-based security finds a wide range of applications, notably to secure delegation of computation to another party. We single out the scenario where a computationally limited party wants to send an encrypted message to a computationally superior party using the RSA protocol, thereby providing another kind of application of decoy ideas in a public-key setting. With typical RSA parameters, decoy-based method of delegation of computation improves the efficiency for the sender by several orders of magnitude.

## 1. Introduction

The idea of relying security on using decoys of the actual private information was introduced in [4] and [5], where the authors employed this idea for secure public-key encryption, as well as for solving Yao's "millionairs' problem", without using any computational assumptions. Some preliminary ideas (coming from group theory) in that direction appeared earlier in [7] (see also [6]).

The justification of decoy-based security is as follows. In all real-life scenarios of information transmission, there is always a nonzero probability of security breach since the adversary can just *guess* whatever piece of (private) information is needed to recover a secret that matters. Therefore, in every real-life scenario there is a probability $p > 0$ for the adversary to get a hold of the secret, and this $p$ has to be accepted by the community as negligible. Thus, in contrast with what is called "theoretical cryptography", in real-life scenarios we tolerate leakage of information as long as the probability for the adversary to get a hold of the actual secret does not exceed the agreed upon value of $p$. A good example is protecting your credit card number. If

Research of the author was partially supported by the NSF grant CNS-1117675 and the ONR (Office of Naval Research) grant N000141210758.

you simply take a credit card out of your wallet in a restaurant and people around you see that it is a MasterCard, say, then this already leaks information about the first 4 digits of your credit card number. However, this does not bring people too close to getting your whole credit card number because the probability of guessing it correctly remains small enough to be neglected.

This suggests the following concept of security. Instead of confronting the adversary with a computationally hard problem that he has to solve to get a hold of a secret plaintext, just give him a pile of possible (not necessarily equally possible) plaintexts ("decoys"), such that the probability for the adversary to pick the "real one" based on all available information does not exceed some $p$ that is commonly accepted as negligible. In other words, "to hide a tree use a forest".

In Section 2, following [5], we describe a possible scenario of decoy-based public-key encryption using a physical medium (e.g. a fiber optics cable). In this scenario, Alice and Bob combine their private keys during the transmission; in particular, they do not take turns to transmit information (as they do in all "traditional" scenarios), but both contribute to the same transmission at the same time.

In Section 3, following [4], we elaborate on another, related, aspect that makes decoy possible. Namely, suppose the adversary can obtain, based on what he observes in the public space, a system of $m$ equations in $n$ unknowns, where the unknowns are Alice's or Bob's private keys or secret messages. Then, if the system is underdetermined, i.e., if $m < n$, chances are there are many different solutions to the system, thus providing for numerous decoys of the actual secret(s). We illustrate this general idea by a specific public-key encryption protocol from [4]. Again, we use simple physics (this time, an electrical circuit) to arrange for an underdetermined system of equations and therefore to provide decoys.

In Section 3.1, we describe one of the elegant (and practical) solutions of Yao's "millionairs' problem" [9] given in [5]. Security of private information in that solution is decoy-based, and decoys are provided, again, by making the adversary face an underdetermined system of equations. (In Yao's "millionairs' problem", Alice and Bob are each other's adversaries.)

In Section 4, following [2], we show how to use the idea of decoy for secure delegation of computation to another party. A popular application of this functionality is outsourcing computations to a cloud, but the method of [2] also yields another interesting application, to the RSA encryption. It is well known that in RSA protocol, to encrypt a plaintext $x$ (which is an element of a ring $\mathbb{Z}_m$), the sender has to raise $x$ to a large power; for strong security, this power can be on the order of 2048 bits. With the "square-and-multiply" method for exponentiation, this will require between 2048 and $2048 \cdot 12$ multiplications, depending on the Hamming distance from the exponent to the nearest power of 2. There is also reduction modulo $m$ involved after almost every squaring, which does not necessarily require

many multiplications in $\mathbb{Z}_m$ (for example, Barrett reduction [1] uses just one multiplication), but then there is some pre-computation involved that brings an extra $\log m$ multiplications, so in the above example we are going to have roughly $2 \cdot 10^4$ multiplications (and about $2 \cdot 10^6$ additions) in $\mathbb{Z}_m$, which may be too much for a small gadget like a car key, or an RFID tag, or an FPGA. With the method of [2], the sender can delegate most of the computation, including the above mentioned pre-computation, to the receiver (who is typically computationally superior) and end up doing under 30 multiplications (and about 3,000 additions), depending on how many decoys the sender uses, which in turn depends on what probability of picking the actual secret from a number of decoys is agreed upon to be negligible. Thus, this method improves the efficiency for the sender by several orders of magnitude, see Section 4.2.

Needless to say, there is a vast amount of literature on "traditional", complexity-based, secure delegation of computation and secure function evaluation, where security is based on employing (allegedly) one-way functions. Secure function evaluation was introduced by Yao in [9], where he described how $n$ participants can securely evaluate any binary circuit. Such a protocol was adapted in [3] by Gennaro, Gentry, and Parno to allow for secure function evaluation between a user and a computational entity (e.g. a cloud) where the computational entity computes the function based on input from the user.

## 2. Combining keys during transmission

As we have mentioned in the Introduction, one possible arrangement where the idea of decoy can be used is where the observable transmission depends on both the sender's and the receiver's private keys (or secret messages). This is probably impossible to arrange in traditional scenarios considered in theoretical cryptography where parties take turns to transmit information. In real life, however, all communications are done using one physical medium or another, and by using a physical medium it is actually possible for two parties to combine their private keys during transmission.

The simplest way to implement this idea is described in [5]. Alice (the sender of a secret message) interprets her secret number $a$ as a physical characteristic relevant to the specific protocol the parties are engaged in. For example, if Alice and Bob use waves (in whatever medium), then $a$ can be the amplitude of Alice's wave. Bob (the receiver) is not just "sitting there" waiting for information from Alice to arrive, but actively participates in the transmission using his private key $b$. Namely, he interprets his secret number $b$ as the same physical characteristic and superposes it with $a$, i.e., he makes sure that the physical characteristic (the amplitude of a wave, say) observable in the public space is $c = a + b$. Since there are many different ways to split $c$ as a sum of two numbers, there are many decoys of the

actual secret $a$, as far as the adversary is concerned. At the same time, the legitimate receiver Bob can easily recover $a = c - b$.

We now give one of the protocols from [5] as an example. Alice and Bob are going to generate waves in a common medium; one can think, for example, of acoustic waves in an "old-fashioned", non-digital phone line. A more modern implementation would involve electromagnetic waves in a fiber optic cable.

Alice and Bob are positioned at points A and B (respectively) of the common medium. Alice wants to transmit to Bob her secret number $A_1 > 0$, which is going to be the amplitude of her wave. Alice and Bob combine their waves (that have the same frequency and phase) to get a wave whose amplitude $A$ is the sum $A_1 + A_2$ of the private amplitudes. Bob then recovers Alice's secret as $A_1 = A - A_2$.

Here is a more formal description of this encryption protocol, see [5].

(1) Alice and Bob publicly agree on the common frequency $\omega$ and phase $\varphi$ of their waves.
(2) Alice starts generating, at her point A, a wave with frequency $\omega$ and phase $\varphi$, while at the same time modulating the amplitude $A(t)$ as a random function of time $t$. Bob, too, starts generating his wave at his point B, with frequency $\omega$ and phase $\varphi$, randomly modulating its amplitude. When Bob starts generating his wave, he tells Alice, publicly, that he is "in business"
(3) Eventually, after getting a confirmation that Bob is "in business", Alice stabilizes the amplitude of her wave at $A_1$, and Bob stabilizes the amplitude of his wave at $A_2$.
(4) After the amplitudes have stabilized, the amplitude of the superposition of Alice's and Bob's waves is $A_1 + A_2$, so Bob recovers Alice's secret $A_1$ as $A_1 = A - A_2$.

Again, the main point is that there are numerous "decoy" possibilities for $A_1$, resulting from the fact that there are many ways to split the public amplitude $A$ as a sum $A = A_1 + A_2$. Thus, different combinations of possible values of the private keys $A_1, A_2$ can result in the same observable quantities in the public space, so that even a computationally unbounded adversary cannot determine the actual secret $A_1$.

## 3. Underdetermined systems of equations

Generalizing the idea from Section 2, we are now going to discuss another, related, aspect that makes decoy possible. Namely, suppose the adversary can obtain, based on what he observes in the public space, a system of $m$ equations in $n$ unknowns, where the unknowns are Alice's or Bob's private keys or secret messages. Then, if $m < n$, chances are there are many different solutions to the system, thus providing for numerous decoys of the actual secret(s). We illustrate this general idea by a specific public-key encryption protocol from [4]. Again, we use simple physics (this time, an electrical

circuit) to arrange for an underdetermined system of equations and therefore to provide decoys.

The initial setup is as follows. Alice wants to send a secret positive number $q_A$ to Bob. We assume that Alice has a private space $U$ (e.g. a private room) where nobody (i.e., neither Bob nor an eavesdropper Eve) can observe her actions. Similarly, Bob has a private space $V$ where nobody can observe his actions. (In the "traditional" scenario, $U$ and $V$ are the parties' personal computers.)

In her private space $U$, Alice has a capacitor $C_1$ of the capacitance $c_A$ with the charge $q_A$, while Bob in his private space $V$ has a capacitor $C_2$ of the capacitance $c_B$ with the charge $q_B$, selected by Bob randomly. These capacitors are connected to form an electrical circuit (see Figure 1), in such a way that the capacitors' plates holding positive charge are connected by one wire, and the plates holding negative charge are connected by another wire. Alice also has a switch that keeps the circuit disconnected until the actual transmission begins, and she has an ammeter to monitor the electric current in the circuit. Bob also has (in his private space) a *rheostat* (i.e., a variable resistor) included in the circuit. This allows him to randomly change the resistance of the whole circuit, and therefore also to change parameters of the electric current during the transmission. Now more formally:

**Alice's (sender's) public key:** capacitance $c_A$

**Alice's secret message:** charge $q_A$

**Bob's (receiver's) long-term private key:** capacitance $c_B$

**Bob's session private key:** charge $q_B$. This private key is selected by Bob randomly before each transmission from Alice.

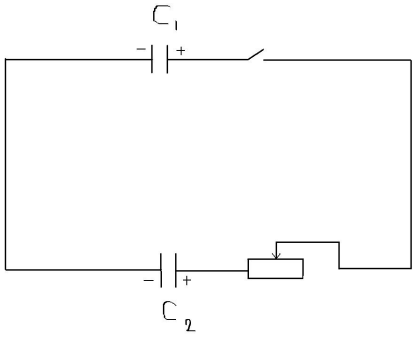


FIGURE 1. Electrical circuit

Now here is the protocol itself:

(1) Alice uses her switch to connect the circuit, thereby starting re-distribution of electric charges between the capacitors $C_1$ and $C_2$. When the re-distribution of the charges is complete, Alice's ammeter shows that there is no current in the circuit, so she disconnects the circuit.

(2) After re-distribution of the charges is complete, let $Q_A$ be the new charge of the capacitor $C_1$, and $Q_B$ the new charge of the capacitor $C_2$. Then:

$$Q_A = c_A \cdot \frac{q_A + q_B}{c_A + c_B}, \qquad Q_B = c_B \cdot \frac{q_A + q_B}{c_A + c_B}, \qquad \frac{Q_A}{c_A} = \frac{Q_B}{c_B}.$$

(3) Bob, who knows $Q_B$, $c_B$, $q_B$, and $c_A$, can now recover Alice's secret $q_A$ from the second formula above: $q_A = Q_B \cdot (1 + \frac{c_A}{c_B}) - q_B$.

Note that in the three equations displayed at Step 2 of the protocol above, there are 5 parameters unknown to the adversary (the only known parameter is $c_A$), which yields many possible solutions for $q_A$. In fact, one of the three equations is redundant (for example, equation 1 follows from the equations 2 and 3), so there are just 2 equations with 5 unknowns to the adversary!

The adversary may attempt to measure the electric current $I$ in the circuit during the transmission and use other laws of physics to try to recover some of the parameters. Relevant laws of physics include Ohm's law $I = \frac{U}{R}$, where $U$ is the voltage and $R$ the total resistance in the circuit. Since right after Alice turns her switch on, the initial voltage $U$ is $\frac{q_A}{c_A} - \frac{q_B}{c_B}$, upon combining these two formulas we get $I = \frac{q_A}{c_A R} - \frac{q_B}{c_B R}$.

This formula introduces two more parameters, at least one of which, the total resistance $R$, cannot be measured by the adversary because some parts of the circuit, including the rheostat with variable resistance, are hidden in the private space.

There are other formulas involving some of the parameters of our electrical circuit, but they all are similar to the formula above in the sense that they introduce new parameters, at least one of which cannot be evaluated by the adversary because it is relevant to properties of those circuit elements that are hidden in either Alice's or Bob's private space.

3.1. **Yao's millionaires' problem.** The "two millionaires problem" introduced by Yao in [9] is: Alice has a private number $a$ and Bob has a private number $b$, and the goal of the two parties is to solve the inequality $a \leq b$? without revealing the actual values of $a$ or $b$, or more stringently, without revealing any information about $a$ or $b$ other than $a \leq b$ or $a > b$.

We note that all known solutions of this problem (including Yao's original solution) use one-way functions one way or another, which means security in all those solutions is complexity-based.

In this section, we offer a very simple solution of Yao's millionaires' problem where security of the parties' private numbers is decoy-based rather than complexity-based. Again, we use a simple physical principle to arrange for both parties' private keys to contribute to the transmission at the same time. Our exposition here follows [5].

Here we have two communicating vessels. One of them, call it $U$, is in Alice's private space, and the other one, call it $V$, is in Bob's private space.

These vessels are connected by a horizontal pipe attached to their bottoms. The shapes of the vessels are part of the parties' private keys.

In the beginning the system is "almost", but not completely, filled with water. Then Alice starts pumping the water *out* of her vessel at the speed of $a$ gallons (or whatever units) per second, while Bob starts pumping the water *in* his vessel at the speed of $b$ gallons per second. The parties are just watching whether the level of water is decreasing or increasing. If it is decreasing, then $a > b$; if it is increasing, then $a < b$.

Note that the final level of water in the system depends not only on $a$ and $b$, but also on the shapes of both vessels. Also, quantities that can be measured outside of, say, Alice's vessel (water pressure, speed of flow, etc.) depend only on the *level* of water in Alice's vessel, whereas Alice's private number $a$ represents the *volume* of water that Alice pumps out of her vessel every second. The relation between this volume and the level of water in Alice's vessel (and therefore the relation between $a$ and quantities that can be measured outside of Alice's vessel) clearly depends on the shape of Alice's vessel, which is unknown to anybody except Alice herself.

Therefore, neither party will be able to determine the other party's number based on the information available to them.

## 4. Secure delegation of computation: evaluating $x^k$

Our exposition in this section follows [2]. Here we show how to use the idea of decoy for secure delegation of computation to another party. A popular application of this functionality is outsourcing computations to a cloud, but the method of [2] also yields another interesting application, to the RSA encryption. Recall that in the RSA scenario, a secret plaintext is an integer $x$ modulo a public integer $m$; the latter is a product of two large primes. The plaintext $x$ is encrypted as $x^k$ modulo $m$, where the exponent $k$ is public. For large $k$ and $m$, computing $x^k$ may be too much for a small gadget like a car key, or an RFID tag, or an FPGA. With the method of [2], the sender can delegate most of the computation to the receiver (who is typically computationally superior). We describe the relevant protocol below.

Thus, suppose Alice (the sender) wants to compute $x^k$ where $x \in \mathbb{Z}_m$ for some large $m$. Suppose that $0 < k < m$ is so large that Alice lacks the computational power to compute it herself. Thus, Alice outsources some of the computations to Bob (the receiver) in the following way:

(1) Alice instructs Bob to select $n$ distinct invertible elements in $\mathbb{Z}_m$, call them $x_1, \dots, x_n$, where $n > \log\log(k)$ and for each $i$, $\frac{m}{2n} \leq x_{i+1} - x_i \leq \frac{m}{n}$, i.e., $x_i$ are sufficiently "sparse". (Without loss of generality, we may assume that $n$ is invertible in $\mathbb{Z}_m$.)
(2) For each $1 \leq i \leq n$, Bob sends $\frac{m}{2n}$, $x_i$, $x_i^k$ and $x_i^{-1}$ to Alice. Alice checks that the sparsity condition $\frac{m}{2n} \leq x_{i+1} - x_i \leq \frac{m}{n}$ is satisfied for each $i$.

(3) Among the elements $x_i$ she got from Bob, Alice chooses $r$ random elements $x_{i_j}$, where $1 \le r \le \log\log(k)$ and $1 \le i_1, \cdots, i_r \le n$. Then Alice computes $y = x x_{i_1}^{-1} x_{i_2}^{-1} \cdots x_{i_r}^{-1}$ and sends $y$ to Bob.
(4) Bob computes $y^k$ and sends it to Alice.
(5) Alice computes $x^k = x_{i_1}^k \cdots x_{i_r}^k y^k$.

Thus, Alice has to do at most $2n$ multiplications in $\mathbb{Z}_m$. The magnitude of $n$ depends on how many decoys for her private $x$ Alice would like to have, see Section 4.1 below.

4.1. **Security.** The security of the above protocol relies on the fact that Bob, or any eavesdropper, will be facing many possibilities for $x$ because of the many possible ways for Alice to build $x$ out of $x_i$ and $y$. This is due to the fact that out of $n$ public potential factors of $x$ only at most $\log\log(k)$ are "true factors". Any choice of less than $\log\log(k)$ of the $x_i$ would correspond to a potential $x$. We call each potential $x$ a decoy. The overall idea is that as Alice chooses larger $n$, the number of decoys increases super-linearly. Hence, for a reasonably large $n$, the probability of guessing $x$ is small enough to be considered negligible.

To compute the total number of decoys, we first count the number of ways to choose $r$ elements from $n$, with repetitions allowed. The formula is well known (see e.g. [8]):

$$\binom{n+r-1}{r} = \frac{(n+r-1)(n+r-2)\cdots(n+1)n}{r!}$$

Note that if we fix $r$ and vary $n$, the above becomes a degree $r$ polynomial in $n$. We now vary $r$ over 1 through $\log\log(k)$. Therefore the total number of decoys is:

$$\sum_{r=1}^{\log\log(k)} \binom{n+r-1}{r}.$$

Again, fixing $k$ and varying $n$ we have that this sum is a polynomial of degree $\log\log(k)$ in $n$, so the number of decoys is $O(n^{\log\log(k)})$. To give some sample values with $k = 2^{2048}$, the number of decoys for $n = 10$ will be 352715, for $n = 20$ it will be about $8 \cdot 10^7$, and for $n = 50$ it will be about $4 \cdot 10^{11}$.

Now let us see why no additional information about $x$ is leaked during the execution of the protocol. First note that the $x_i$ are chosen (by Bob) independently of $x$, so no information about $x$ can possibly be leaked at this step. Only at Step 3, when Alice sends $y$ to Bob, can any information that pertains to $x$ be possibly leaked. However, since $x$ is unknown to Bob or any eavesdropper and is (supposedly) distributed uniformly in $\mathbb{Z}_m$ to begin with, $y$ is also distributed uniformly in $\mathbb{Z}_m$, and therefore cannot leak any information about $x$.

4.2. **Complexity.** We now show that by using the protocol in Section 4, Alice does logarithmically many multiplications compared to what she would do if she were computing $x^k$ by herself, without Bob's help. To compute $x^k$, the fastest method Alice can use on her own is the "square and multiply" method, in which case she would have to perform between $\log(k)$ and $\log(k) \cdot \log\log(k)$ multiplications, depending on $k$. This is without taking into account the complexity of reductions modulo $m$.

With Bob's help, Alice performs approximately $\log\log(k)$ multiplications at Step 3 and then another $\log\log(k)$ multiplications at Step 5. Therefore, she performs logarithmically many multiplications compared to what she would without Bob's help, i.e. with this protocol she can delegate most of the work to Bob.

**Example:** Let $k \sim m \sim 2^{2048}$. To compute $x^k$ for some $x \in \mathbb{Z}_m$ using the "square and multiply" method, one would need to do between $\log(k) \sim 2048$ and $\log(k) \cdot \log\log(k) \sim 20,000$ multiplications, i.e., about $10,000$ multiplications on average (not counting reductions modulo $m$). By using the above protocol with $n = 20$, say, Alice would instead only need to do under 40 multiplications. Note that with $n = 20$, the number of decoys would be roughly $10^8$ (see Section 4.1), so the probability of guessing the actual $x$ is approximately $10^{-8}$, which can be considered negligible for most practical purposes.

In [2], the authors also reported on computer experiments comparing the actual performance time, on a typical desktop computer, of their method of delegating computation versus the standard "square-and-multiply" method. According to these experiments, computation of $x^k$ (using parameters as in the above example) with their method is about 200 times faster (for the sender). See [2] for more details.

The total time the RSA-2048 encryption took in their experiments was just under one second (on average), which may not be super fast, but is still practical. The point of using their method is that the sender's share in this total time is just about 0.0003 seconds.

Finally, we mention that computation of arbitrary monomials can also be done by way of delegating it to a computationally superior party, see [2] for details.

## References


[1] P. Barrett, *Implementing the Rivest, Shamir and Adleman public key encryption algorithm on a standard digital signal processor*, in: Advances in Cryptology CRYPTO 1986, Lecture Notes Comp. Sc. **263** (1986), 311-323.

[2] B. Cavallo, D. Kahrobaei, V. Shpilrain, *Decoy-based secure delegation of computation, with application to RSA encryption*, preprint.

[3] R. Gennaro, C. Gentry, and B. Parno, *Non-interactive verifiable computing: Outsourcing computation to untrusted workers,* in: Advances in Cryptology – CRYPTO 2010, Lecture Notes Comp. Sc. **6223** (2010), 465-482.

[4] D. Grigoriev and V. Shpilrain, *Secure information transmission based on physical principles*, in: UCNC 2013, Lecture Notes Comp. Sc. **7956** (2013), 113-124.

[5] D. Grigoriev and V. Shpilrain, *Yao's millionaires' problem and decoy-based public key encryption by classical physics*, J. Foundations Comp. Sci. **25** (2014), 409-417.

[6] A. G. Myasnikov, V. Shpilrain, and A. Ushakov, *Non-commutative cryptography and complexity of group-theoretic problems*, Amer. Math. Soc. Surveys and Monographs, 2011.

[7] D. Osin and V. Shpilrain, *Public key encryption and encryption emulation attacks*, in: Computer Science in Russia 2008, Lecture Notes Comp. Sc. **5010** (2008), 252-260.

[8] R. P. Stanley, *Enumerative combinatorics.* Cambridge University Press, 2011.

[9] A. C. Yao, *Protocols for secure computations.* In: Proceedings of the 23rd Annual Symposium on Foundations of Computer Science, pp. 160-164. IEEE Computer Society, 1982.

The City College of New York and CUNY Graduate Center
*E-mail address*: shpil@groups.sci.ccny.cuny.edu