Insecurity Against Selective-Opening Attacks: Some Key Ideas

Selected Paper: *Standard Security Does Not Imply Security Against Selective Opening* by Bellare, Dowsley, Waters, and Yilek

James Mouradian

March 18, 2012

Introduction

Many of the standard notions of security we have examined quarter with respect to two-party communication may fail to capture notions of security in unusual settings. One such setting, not unrealistic in multi-party protocols, is the setting in which a receiver receives n individual messages from n distinct senders, all of which are encrypted and committed to their values. At some point, the receiver may pick a subset of the messages to be opened, e.g., have their contents decrypted and verified, before the protocol continues. An adversary may perform what is known as a selective-opening attack (SOA) to corrupt this subset of messages, obtaining both the messages it contains *and* the random parameters (hereafeter referred to as coins) used to encrypt the messages. The authors demonstrate that an adversary who obtains both the messages *and* coins is able to, with high advantage, produce an output which it would not be able to replicate without receiving the coins. Concisely phrased, standard security does not imply security against selective-opening.

Security against selective-opening attacks can be misleadingly thought of in terms of the following question: "If an adversary obtains the plaintexts and coins of some subset of messages, are the remaining messages still secure?" In the attack presented, the adversary does *not* satisfy a relation that explicitly reveals information about the un-opened messages; however, the adversary is still able to accomplish something it was unable to without accessing the coins. This highlights the strength of SOA-secure schemes that the authors provided after writing this paper, presented in class by Robert.

A wide variety of encryption schemes leak the coins used for encryption. The authors provide an ElGammal-based example. Other examples include using RSA with OAEP, or CBC-mode with a prepended IV. The commonality of such schemes highlights the reality of SOA-insecurity, and the necessity for the authors' SOA-secure schemes.

SOA-C Insecurity of Hiding and Binding Schemes

First consider an ideal world: we would like to establish a protocol in which n senders each commit to a single message at a given point in time; from that point forward, they are unable to change the message they would like to send. At a future point, the receiver of these messages selects which subset of messages to receive. The receiver then receives only these messages, about which she had no prior knowledge, and is confident that the senders did not change their messages. We will represent this ideal case with a *simulator*. We will say an adversary wins a game if it can produce an output that satisfies some boolean relation that the simulator cannot.

In a real-world scenario, we will use an encryption scheme to try to achieve the goals stated in the ideal case. After considering the ideal scenario, we might at first wish to ensure our encryption scheme is both *hiding* and *binding*. A scheme is *hiding* if an adversary cannot learn anything about the messages before opening them, and a scheme is *binding* if an adversary cannot give two distinct plaintexts for a given ciphertext. Ironically, the authors show that any protocol which is both hiding and binding is SOA-C insecure, assuming only the existence of collision-resistant hash functions.

In a real-world protocol, let there be n senders and a single receiver. The senders will independently determine their messages m and coins r to deterministically encrypt with an encryption scheme $\mathcal{E}$ a ciphertext $c = \mathcal{E}(m; r)$. Let $\mathbf{m}$ denote a vector of messages, $\mathbf{r}$ a vector of coins, and $\mathbf{c}$ a vector of ciphertexts such that $\mathbf{c} = \mathcal{E}(\mathbf{m}; \mathbf{r})$. Let $\mathbf{x}[i]$ denote the i-th element of the vector $\mathbf{x}$. Our adversary will pick some subset I of the numbers $\{1, 2, ..., |\mathbf{c}|\}$, and corrupt the set of ciphertexts $\{\mathbf{c}[i] : i \in I\}$, receiving the messages $\{\mathbf{m}[i] : i \in I\}$ and coins $\{\mathbf{r}[i] : i \in I\}$. Our simulator would have received only the set of messages, not the set of ciphertexts or coins.

Our adversary mounts an attack as follows. Let $H : \{0,1\}^* \to \{0,1\}^h$ be a collision-resistant hash function which outputs a vector $\mathbf{b}$ of bits, and let $n = 2h$. Upon receiving the ciphertext vector $\mathbf{c}$, the adversary concatenates the ciphertexts into a single string, and then computes $\mathbf{b} = H(\mathbf{c})$. The adversary then selects the subset I of indeces to corrupt as $I = \{2j - 1 + \mathbf{b}[j] : 1 \leq j \leq h\}$. It should be obvious that knowing the subset I of n is equivalent to knowing the output of the hash function. The adversary outputs a value $w = \mathbf{c}||\langle\{\mathbf{r}[i] : i \in I\}\rangle$. The adversary wishes to satisfy two constraints on its output: the *opening constraint*, or that $\mathbf{c} = \mathcal{E}(\mathbf{m}; \mathbf{r})$, and the *hash constraint*, or that $I = \{2j - 1 + \mathbf{b}[j] : 1 \leq j \leq h\}$ for $\mathbf{b}[1] \ldots \mathbf{b}[h] = H(\mathbf{c})$. The adversary wins if both these constraints are met. It should be obvious that the adversary is able to always meet both of these constraints for any $\mathbf{c} = \mathcal{E}(\mathbf{m}; \mathbf{r})$ after successfully performing an SOA. The simulator should attempt to, given only $\{\mathbf{m}[i] : i \in I\}$, produce an output that matches these constraints.

Suppose the simulator were to attempt to satisfy the opening constraint. Satisfying the opening constraint is easy - the simulator must simply produce a vector of coins $\mathbf{r}$, and encrypt the messages $\{\mathbf{m}[i] : i \in I\}$ using these coins. The simulator then fills in remaining gaps for $\{\mathbf{c}[i] : i \notin I\}$ with any bits it so chooses. However, if H is collision-resistant, this is a task at which the simulator gains negligible advantage.

Suppose instead the simulator were to prioritize satisfying the hash constraint. Then, it might attempt to construct ciphertext vectors $\mathbf{c}$ such that $I = \{2j - 1 + \mathbf{b}[j] : 1 \leq j \leq h\}$, where $\mathbf{b} = H(\mathbf{c})$. If H is collision-resistant, finding any such $\mathbf{c}$ would be difficult. However, even if the simulator found any such $\mathbf{c}$, meeting opening constraint would be difficult. The simulator would, given its arbitrarily chosen ciphertexts, attempt to find coins $\mathbf{r}$ that, when paired with messages $\mathbf{m}$ ensure this new $\mathbf{c} = \mathcal{E}(\mathbf{m}; \mathbf{r})$. This is difficult because since $\mathcal{E}$ is binding, the simulator should not be able to construct messages and coins of its choice to produce a given ciphertext.

The authors formalize the difficulty of finding hash collisions and violating the binding property of an encryption scheme, and show that the ability for a simulator to produce an output w which satisfies both the binding and hash constraints is negligible. Since our adversary can always meet both these constraints, its advantage in this context is close to 1.

Conclusions and Beyond

The authors successfully demonstrate that no hiding and binding encryption scheme is secure against a selective-opening attack. Many hiding and binding encryption schemes offer the adversary the coins used for encryption, so SOA-insecurity of real-world schemes is a prevalent problem that the authors felt the need to address.

In addition to the notion of SOA-C security the authors discuss, they also discuss the notion of SOA-K security. In the SOA-K notion, a sender sends n messages to n distinct receivers. An adversary who performs the attack receives not only the subset I of messages, but also the decryption keys used to decrypt those messages. Adversaries in this context gain high advantage by using an extremely similar procedure: selecting a subset of messages to corrupt using a collision-resistant hash function, and outputting a ciphertext and the keys used to decrypt the subset of messages.

The authors follow up this contribution by offering schemes which *are* SOA-secure. Such schemes are lossy and non-committing, and the authors prove that the schemes are SOA-secure for these reasons.