Key Agreement from Close Secrets over Unsecured Channels

Winter 2010

Andreas Keller

Contens

1. Motivation
2. Introduction
3. Building Blocks
   - Extractor
   - Secure Sketches
   - (MAC) message authentication codes
   - Authentication
4. Protocol
5. Implementation Results

Motivation

There are mainly two types of cryptographic security, namely:

- computational and

- information-theoretic cryptographic security

Motivation

In the present paper, we take a step towards making unconditional security more practical by showing that such a private key can be generated, by communication over a completely insecure channel, from an arbitrarily weakly secret key.

Introduction

We build on the results of

"Unconditionally secure asymmetric cryptography." and

"Robust fuzzy extractors and authenticated key agreement from close secrets."

by proposing a protocol that is efficient for both parties and has both lower round complexity and lower entropy loss than the protocol of "Unconditionally secure asymmetric cryptography."

Introduction

Protocol

We start with an authentication sub-protocol *Auth* that achieves the following:

- it allows Alice to send to Bob an authentic (but nonsecret) message M of length λ_M bit-by-bit in $2\lambda_M$ messages using the secret w that is common to Alice and Bob

Alice and Bob use this sub-protocol in order to agree on a key k as follows:

- they use *Auth* to get an extractor seed s from Alice to Bob, and then extract k from w using s.

Introduction

Protocol

We modify this protocol by using *Auth* to authenticate a *MAC key* instead of an *extractor seed*.

The MAC key is used to authenticate the *extractor seed*.

This seems counterintuitive, because *Auth* reveals what is being authenticated, while MAC keys need to remain secret. The insight is to use the MAC key before Auth begins.

Introduction

Protocol

Our modification is beneficial for three reasons:

- *MAC keys* can be made shorter than extractor keys, so *Auth* is used on a shorter string, thus reducing the *entropy loss*.
- allows us to use the same *MAC key* to authenticate not only the extractor seed *s*, but also the error-correction information (the so-called "secure sketch" of *w*) in the case Bob's *w* is different from Alice's *w*.
- because there are MACs that are secure even against (limited) key modification, we can lower the security parameters in *Auth*, further increasing efficiency and reducing entropy loss.

Building Blocks

Extractors

Definition: Let $\mathsf{Ext} : \{0,1\}^n \rightarrow \{0,1\}^\ell$ be a polynomial time probabilistic function which uses r bits of randomness. We say that *Ext* is an efficient (n, m, ℓ, ϵ)-*strong extractor* if for all min-entropy *m* distributions *W* on $\{0,1\}^n$, statistical distance $\mathrm{SD}\left((\mathsf{Ext}(W;X), X), (U_\ell, X)\right) \leq \epsilon$ where *X* is uniform on $\{0,1\}^r$.

Building Blocks

Secure Sketches

allow the error-correcting information that Alice sends to Bob in the first message as a secure sketch.

Secure sketches provide two algorithms:

- "generate" (Gen) that takes an input *w* and produces a sketch *P* and
- "recover" (Rec) that outputs *w* from the sketch *P* and any *w'* sufficiently close to *w*.

Their security guarantees that some entropy remains in *w* even given *P*.

Building Blocks

Secure Sketches

In our case we do not need a full recovery of the original *w*: we will be satisfied if both *Gen* and *Rec* produce some string *R* that preserves some of the entropy of *w*.

Our new primitive is like a fuzzy extractor. We call the primitive a weakly robust fuzzy conductor because it conducts entropy from *w* to *R* and is robust against active attacks on *P*.

Building Blocks

MAC

We use a One-time MACs allow information-theoretic authentication of a message using a key shared in advance.

MAC Construction.

We will use the following standardMAC technique:

View the key k as two values, a and b, of $\lambda\sigma$ bits each. Split the message M into c chunks $M_0, \ldots, M_{c-1}$, each $\lambda\sigma$ bits long, and view these as coefficients of a polynomial $\tilde{M}(x) \in F_{2^{\lambda\sigma}}[x]$ of degree $c-1$.

Then $\mathrm{MAC}_k(M) = a\,\tilde{M}(a)+b$. This is a $\lambda_M/\lambda_\sigma\, 2^{-\lambda\sigma}$ -secure message authentication code.

Building Block

MAC

This construction has two properties that are particularly important to us:

- First, its key length is close to optimal.
- Second, it is secure even when the adversary knows something about the key, with security degrading according to the amount of information adversary knows.

Authentication

Alice and Bob share a string R. Alice wishes to authentically send Bob $M = M_1 \dots M_{\lambda_M}$ of λ_M bits. The value λ_M and the number of ones in M is known to Bob.
For i = 1 to λ_M:

1. Alice sends Bob challenge $x_i \in_r \{0,1\}^q$.

2. Bob receives x'_i, sends $b'_i = \mathsf{Ext}(R; x'_i)$, and challenge $y'_i \in_r \{0,1\}^q$

3. Alice receives b_i, y_i, verifies that $b_i = \mathsf{Ext}(R; x_i)$ and aborts if not.
   She sends $(1, a_i = \mathsf{Ext}(R; y_i))$ if $M_i = 1$, and $(0, \perp)$ otherwise.

4. Bob receives b'_i, a'_i aborts if $b'_i = 1$ and $a'_i \neq \mathsf{Ext}(R; y'_i)$
   and accepts otherwise.
   If $i = \lambda_M$, Bob verifies that the number of ones in the received string match the expected number of ones; aborts otherwise.

Note that step 3 and 4 of each iteration are combined with steps 1 and 2, respectively, of the next iteration.

Building Block

Authentication

The authentication protocol allows two parties who share the same string R to authenticate a message M, even if R has very little entropy.

We assume that *Ext* is an average-case extractor that takes seeds of length q, and outputs $L + 1$-bit strings that are 2^{-L-1}- close to uniform as long as the input has sufficient entropy h.

For our purposes, it suffices to assume that the length of M and the number of ones in it (i.e., its Hamming weight wt(M)) are known to Bob.

Building Block

Authentication

The intuition for the security of this protocol is that Eve can not answer a random query x_i or y_i with probability greater than $2-L$ because of the entropy of the answers, and hence can neither remove zero bits nor insert one bits .

She can insert zero bits and change zeros to ones, but that is taken care of by the assumption that Bob knows λ_M and Hamming weight wt(M).

Protocol

We propose the following privacy amplification protocol, in which Alice starts with w and Bob with w' such that $\mathrm{dis}(w,w') \leq \eta$.

1. Alice generates a random MAC key k_1 and extractor seed s_1, computes $(R,P) \leftarrow \mathrm{Gen}(w)$,

$\sigma_1 = \mathrm{MAC}k_1(s_1, P)$, and sends $((s_1, P), \sigma_1)$ to Bob.

2. Alice initiates the message authentication protocol *Auth* for the message k_1 using R as the shared secret value.

3. Bob receives $((s_1, P), \sigma_1)$, and computes $R = \mathrm{Rep}(w, P)$. He responds to Alice's *Auth* protocol, using the string R as the shared secret value.

Protocol

4. Upon completion of Alice's side of Auth (if she has not yet rejected), Alice

- extracts $k_2 = \mathrm{Ext}_1(R; s_1)$;
- generates a random seed s_2;
- sends Bob s_2 and $\sigma_2 = \mathrm{MAC}k_2(s_2)$;
- outputs her final key $\mathrm{k_A} = k_3 = \mathrm{Ext}_2(R; s_2)$

Protocol

5. Upon completion of Bob's side of the *Auth* with the received message k'_1, and receipt of s'_2, σ'_2 from Alice, Bob:

- verifies the first MAC, $Verify\_k'_1\ ((s'_1, P'), \sigma'_1)$ (if fail, rejects);
- computes the key for the second MAC, $k'_2 = \mathrm{Ext}_1(R'; s'_1)$;
- verifies the second MAC, $Verify\_k'_2\ (s'_2, \sigma'_2)$ (if fail, rejects);
- outputs his final key $k_B = k'_3 = \mathrm{Ext}_2(R'; s'_2)$

Security of this protocol

The intuition behind the security of this protocol is in the ordering of events.

1. Alice authenticates a message (s_1, P) to Bob using a MAC with a truly random key k_1 which is unknown to Eve.
2. after she is sure that Bob has received the message and the tag σ, Alice transmits k_1 to Bob using the authentication protocol.
3. Alice adds a level of indirection, using k_2 as a key to authenticate another extractor seed s_2, which is then used to extract the output key.

Analysis

The security parameter for our protocol is L.

For any string x we use λ_x to denote the length of the x and h_x to denote its entropy $H_\infty(x)$

Robustness: We can view the protocol as consisting of two phases.

- Phase 1: Agreeing on k_2 from close secrets w,w'
- Phase 2: Using k_2 to agree final string k_3

Analysis

Security of Phase 1.

Suppose Eve succeeds in an active attack against Phase 1, i.e., $k_2 \neq k'_2$. There are two possibilities:

1. $k_1 = k'_1$ (Eve does not attack protocol Auth). Therefore, in order for $k_2 \neq k'_2$, either $s_1 \neq s'_1$ or $P \neq P'$. Because Bob verifies the first MAC, Eve needs to come up with a valid $((s'_1, P'_1), \sigma'_1)$, P

which she has to do when she forwards Bob his very first message.

Analysis

2. $k_1 \neq k'_1$: In this case, Eve has to authenticate $k'_1 \neq k_1$, using *Protocol Auth* in order to succeed. Therefore, her chances of success in this case are bounded by her chances of succeeding in an active attack against *Auth*. Again, if we run *Auth* on the security parameter $L + 1$, we can show that $Pr[auth] \leq 2^{-L}$.

Analysis

Security of Phase 2.

The key $k_2 = \mathrm{Ext}(R, s)$ agreed upon by the parties at the end of Phase 1 is used in Phase 2 to authenticate a extractor seed s_2 using the single message MAC.

- However, the authentication protocol of Phase 1 gives Eve the ability to query the parties and get some information about $Ext(w, s_1)$, decreasing the entropy of k_2. Knowing that this decrease will be no more than the amount communicated about R during Phase 1 , we will set the length of k_2 to be twice that plus $2L+2log\ \lambda s_2/L$ to get the desired 2^{-L} security for the second MAC.

Implementation Results

We implemented our protocol using Shoup's NTL.

The protocol was tested for L = 80 and n = 100, 000 on

a LAN with Alice and Bob running on a 2.4Ghz Intel Pentium and a WAN with Alice running on a 2.4Ghz Intel Xeon instead.

The running times over a WAN and LAN were nearly the same, both less than 5 seconds.

Of the total running time, approximately 1.5 seconds were spent by each party on computation and

an additional 1 second was spent in total communication costs.