Preliminary draft.

The Security of Chaffing and Winnowing

Mihir Bellare[*] Alexandra Boldyreva[†]

January 2000

Abstract

This paper takes a closer look at Rivest's chaffing-and-winnowing paradigm for data privacy. We begin with a *definition* which enables one to determine clearly whether a given scheme qualifies as "chaffing-and-winnowing." We then analyze Rivest's schemes to see what quality of data privacy they provide. His simplest scheme is easily proven secure but is ineffient. The security of his more efficient scheme —based on all-or-nothing transforms (AONTs)— is however more problematic. It can be attacked under Rivest's definition of security of an AONT, and even under stronger notions does not appear provable. We show however that by using a OAEP as the AONT one can prove security. We also present a different scheme, still using AONTs, that is equally efficient and easily proven secure even under the original weak notion of security of AONTs.

Keywords: Message authentication, symmetric encryption, proofs of security, cryptographic policy.

[*]Dept. of Computer Science & Engineering, University of California at San Diego, 9500 Gilman Drive, La Jolla, California 92093, USA. E-Mail: mihir@cs.ucsd.edu. URL: http://www-cse.ucsd.edu/users/mihir. Supported in part by NSF CAREER Award CCR-9624439 and a 1996 Packard Foundation Fellowship in Science and Engineering.

[†]Dept. of Computer Science & Engineering, University of California at San Diego, 9500 Gilman Drive, La Jolla, California 92093, USA. E-mail: aboldyre@cs.ucsd.edu. URL: http://www-cse.ucsd.edu/users/aboldyre. Supported in part by grants of first author.

Contents

1 Introduction

Rivest presents a number of methods to achieve data privacy based on a paradigm he calls "chaffing and winnowing" [8]. In this paper we

— Provide a *definition* of chaffing and winnowing
— Assess whether the schemes of [8] can be *proven* to meet standard data-privacy goals, and, if so, under what kinds of *assumptions* on the underlying primitives
— Suggest more efficient schemes and analyze their security.

Let us first provide some background and motivation, and see what are the basic questions. Then we discuss our contributions in more detail.

1.1 Background, motivation and questions

Chaffing and winnowing uses a message authentication code (MAC) to provide privacy. This by itself is hardly novel, particularly since Rivest notes that in order to have privacy the MAC must be a pseudorandom function— there are many well-known ways to use a pseudorandom function to provide privacy. The interest of chaffing and winnowing arises from the particular manner in which the MAC is used, which is roughly the following. Each data block is authenticated so that one has a sequence of data-MAC pairs. Then "chaff" is interspaced, this consisting of pairs, each being a block with a random tag. The receiver can discard blocks with invalid tags —this is called "winnowing"— thereby recovering the data. (Within this framework, many specific methods are possible.) Privacy requires that it be computationally infeasible for an adversary to tell valid MACs from random tags. (But is also very sensitive to the manner in which chaff is interspaced.) Rivest argues that the use of the MAC here stays within its functionality as an authentication mechanism, and thereby makes moot a policy that restricts "encryption" while allowing authentication.

Chaffing-and-winnowing has received a lot of attention on the political front, but little on the technical front barring that in the initial paper. It merits more serious attention from cryptographers. An immediate reason is that a better technical understanding leads to a better understanding of the implications for the debate on cryptographic policy. The more important reason however is foundational. As Rivest notes in introducing his idea, there are very few paradigms for cryptography's main goal, namely data privacy. A new paradigm such as the one he presents should to be explored in order to assess its potential.

The description and examples in [8] suffice to get the gist of the idea, but as we consider more complex mechanisms it is sometimes difficult to decide whether they obey the "rules of the game." A definition is needed to settle such questions, and also for rigorous security analyses. Accordingly, we begin there.

Chaffing and winnowing purports to provide data-privacy. A basic question is about the quality of privacy that it can provide. Specifically we want to know how it compares to standard mechanisms such as modes of operation of a block cipher, which have been proven to meet strong, well-defined notions of privacy under appropriate assumptions [1]. Can chaffing and winnowing based schemes provide the same level of privacy, and, if so, can this be proven, and under what assumptions?

The final question we want to address is the cost of the method. How cost-effective can chaffing and winnowing be, especially compared to standard encryption mechanisms?

1.2 Defining chaffing and winnowing

The security goal of a chaffing and winnowing scheme is to provide privacy in a symmetric setting. Accordingly, from the security point of view, it is —in fact, must be— treated simply as a symmetric encryption scheme. There is some "encryption" process that takes a message and creates a "ciphertext", and some "decryption" process that takes the ciphertext and recovers the message, both operating under a common secret key. (This is the key for the MAC.) These processes are not implemented in "usual" ways, but, abstractly, they must exist, else it is moot to talk of achieving privacy. Once this is understood, security can be measured using any of several well-known notions in the literature. (We adopt the simplest, namely the "find-then-guess" notion of [1], which is the most direct extension to the symmetric case of the notion of indistinguishability of [7].)

Thus what defines chaffing and winnowing as a "notion" is not some novel security property but rather a novel set of restrictions on the processes (namely encryption and decryption) directed at achieving a standard security property (namely data privacy). The crux of the definition is to pin down these restrictions. We view a chaffing and winnowing based encryption scheme as arising by the use of an *authentication to privacy transform* (ATPT) over a *MAC-based authentication channel.*

The channel captures the manner in which the parties have access to the MAC function $\mathsf{MAC}(K, \cdot)$. An application on the sender side can pass data down to be MACed, thereby creating a packet (data-MAC pair) which is transmitted over the channel. (The application has no direct access to the $\mathsf{MAC}(K, \cdot)$ function let alone to the underlying key K.) At the receiving end, packets with invalid MACs are dropped and the data from valid packets is passed up to the receiving application. (The latter sees no MACs and does not even know of the existence of the dropped packets.) See Figure 1 and Definition 3.1.

The ATPT consists of three algorithms whose important feature is that they are all entirely *keyless*. The sending application applies a MakeWheat algorithm to the plaintext to turn it into a sequence of data blocks to be passed to the authentication channel. (In one of Rivest's schemes, this algorithm uses an all-or-nothing transform.) An AddChaff procedure is responsible for interspacing chaff packets into the stream of valid packets output by the authentication channel. Finally the receiving application applies a Recover algorithm to the received data blocks to get back the plaintext. See Definition 3.2.

We stress again that the algorithms in the ATPT are keyless, so that on their own they cannot be used to provide privacy. The chaff and winnow based encryption scheme is realized by coupling these algorithms with the authentication channel. This is illustrated in Figure 2.

This definition can help clarify the contribution of chaffing and winnowing to the debate on cryptographic policy by providing a means to evaluate whether a particular method qualifies as "legal encryption based on authentication." If one scheme meeting the definition qualifies, so do the rest, even if their implementation is more complex.

1.3 Security of Rivest's schemes

Rivest notes that his first few examples will not provide a high level of privacy. (In particular they will not meet a notion of privacy such as find-then-guess.) The first serious candidate is the bit-by-bit scheme.

BIT-BY-BIT SCHEME. Here the MakeWheat procedure splits the plaintext into bits and appends a counter or nonce to each bit. These data blocks are MACed. The AddChaff procedure inserts, for every valid packet, an invalid packet with the opposite bit value and an appended nonce, together with a random value for the tag. We prove that this scheme provides privacy in the find-then-guess sense assuming the MAC is a pseudorandom function. The concrete security analysis is provided in Theorem 4.2.

This indicates that chaffing and winnowing can provide privacy of as high a quality as standard encryption schemes, and furthermore with provable guarantees based on the same assumption — namely a pseudorandom function— used to prove the security of popular block cipher modes of operation. There is however a high cost in bandwidth: two nonces and two tags are needed per bit of plaintext.

Scattering schemes. In order to reduce the bandwidth, Rivest suggests an alternative paradigm. First apply an all-or-nothing transform (AONT) [9] to the plaintext. (This is a keyless, invertible transform with the property that inversion is hard if any block of the output is missing.) Each block of the output of the AONT is MACed, resulting in a stream of valid packets. Then s' chaff packets are inserted into random positions in this stream. Intuitively, an adversary must guess the positions of all s' chaff packets in order to decipher. (Accordingly it is suggested that security will be provided for a value of s' that does not depend on the length of the plaintext, eg. $s' = 128$, so this method is cost-effective for long plaintexts.) Upon closer examination, however, the security provided by this paradigm is unclear. We note first that under the original definition of an AONT provided in [9] the scheme is insecure. (We show that there are example AONTs that meet the definition of [9] but for which there are attacks compromising the privacy of the chaffing and winnowing scheme.) It is natural to then try to use Boyko's stronger definition of security for an AONT [5]. In that case the analysis is inconclusive: the stronger property of an AONT still does not appear to suffice to prove security of the chaffing and winnowing scheme, but neither do we exhibit a counter-example that confirms this. We would prefer a mechanism for which a proof is possible.

Scattering with OAEP. The above-mentioned analyses indicate that in general an AONT seems neither necessary nor sufficient as the initial transform to provide privacy of the scattering based chaffing and winnowing scheme. We show however that if the OAEP transform of [4] is used as the AONT, then privacy can be proved. (This, like all security proofs involving OAEP, is in a random oracle model [3]). The concrete security analysis provided in Theorem 5.2 supports the intuition regarding the scattering scheme provided in [8]— the probability of breaking the scheme is inversely proportional to $\binom{s+s'}{s}$ where s is the number of blocks in the output of OAEP and s' is the number of chaff blocks.

Note that OAEP has been shown to be a secure AONT [5], but given the above we cannot exploit this here. Instead, our proof is direct, based on techniques from [4, 5].

1.4 New schemes

Recall that the motivation for the scattering scheme was to reduce bandwidth relative to the bit-by-bit scheme. The security is provable in a specific case, namely when the AONT is OAEP. We would prefer a method that could be more generally instantiated and proven secure, and also had the potential to avoid the random oracle model in analyses.

We point out that there is indeed an alternative method that is simpler, can be proven secure, and is equally cost-effective. It too makes use of AONTs, but requires them only to meet the weak security requirement of [8] as opposed to the strong one of [5]. (This makes it more likely that random oracles are not required for a secure instantiation.) Apply the AONT to the plaintext as before. Rather than scattering chaff into the output blocks, however, simply treat a prefix of this output as the plaintext for the bit-by-bit chaffing and winnowing scheme and apply the latter. In other words, we are suggesting that one can use a general paradigm: apply an AONT to the plaintext and then encrypt a prefix of the output of the AONT. Theorem 6.2 provide a concrete security analysis of this paradigm at a general level, and Corollary 6.3 summarizes the security of the final chaffing and winnowing scheme. (It is surprising that this simpler method was not suggested earlier in this setting. The reason could be that it was unclear whether it really met the "rules of the game" in terms of being a chaffing and winnowing scheme. Having a definition enables us to answer this, and in fact we show that it does meet the definition.)

1.5 Is chaffing and winnowing "encryption"?

We view chaffing and winnowing schemes as (special kinds of) symmetric encryption schemes, the key for encryption and decryption being that of the MAC function. This might at first seem to contradict Rivest's view [8]. He says that the process of chaffing and winnowing is "not encryption" and that there is no "decryption key." These views are not at odds with each other; the difference is purely in terminology. We are using the technical terminology of cryptographers which is more suited to security analyses, while Rivest uses the terminology of cryptographic policy discussion. (The convention in modern cryptography, which we are following here, is to use the term "encryption scheme" for any mechanism whose goal is to provide privacy. Under this convention, the key for the MAC is, by definition, a decryption key, since it enables recovery of the plaintext from the ciphertext. In cryptographic policy, "encryption" seems to refer to certain mechanisms rather than a goal. Actually, exactly what it refers to is unclear, which is part of the point made in [8].)

2 Symmetric encryption, PRFs and AONTs

We wish to assess the (concrete) security of certain chaff-and-winnow based confidentiality procedures when viewed as symmetric encryption schemes, assuming the underlying MAC is a pseudorandom function. To do this we need to briefly recall formal notions of security for symmetric encryption and PRFs. We also need security definitions for all-or-nothing transforms which are used in some schemes.

SYMMETRIC ENCRYPTION. A symmetric encryption scheme $\mathsf{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ consists of a (randomized) key generation algorithm $\mathcal{K}$ (returning a key K), a (randomized or stateful) encryption algorithm $\mathcal{E}$ (taking K and a message $M \in \{0,1\}^*$ to return a ciphertext C) and a decryption algorithm $\mathcal{D}$ (taking K and a ciphertext and returning a message). We require that $\mathcal{D}_K(\mathcal{E}_K(M)) = M$ for all $M \in \{0,1\}^*$. In the "find-then-guess" model [1] an adversary is given an oracle for encryption under key K and wins if it can find two equal-length messages whose ciphertexts it can later distinguish. Below we associate to any adversary an "advantage" which measures its winning probability, and then use as security measure of the scheme the maximum possible advantage subject to stated resource restrictions on the adversary.

Definition 2.1 [Find-then-guess security of encryption, [1]] Let $\mathsf{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a symmetric encryption scheme. For an adversary A and $b = 0, 1$ define the experiment

Experiment $\mathbf{Exp}^{\mathrm{priv}}_{\mathsf{SE}}(A, b)$
  $K \xleftarrow{R} \mathcal{K}$; $(M_0, M_1, St) \leftarrow A^{\mathcal{E}_K(\cdot)}(\mathsf{find})$; $C \leftarrow \mathcal{E}_K(M_b)$; $d \leftarrow A^{\mathcal{E}_K(\cdot)}(\mathsf{guess}, C, St)$
  `Return` d

It is mandated that $|M_0| = |M_1|$ above. Now define the *advantage* of A and the *advantage function* of the scheme respectively, as follows:

$$\mathbf{Adv}^{\mathrm{priv}}_{\mathsf{SE}}(A) = \Pr\left[\mathbf{Exp}^{\mathrm{priv}}_{\mathsf{SE}}(A, 0) = 0\right] - \Pr\left[\mathbf{Exp}^{\mathrm{priv}}_{\mathsf{SE}}(A, 1) = 0\right]$$

$$\mathbf{Adv}^{\mathrm{priv}}_{\mathsf{SE}}(t, q, \mu) = \max_{A}\left\{\mathbf{Adv}^{\mathrm{priv}}_{\mathsf{SE}}(A)\right\}$$

where the maximum is over all A with "time-complexity" t, making at most q encryption oracle queries, these totalling at most μ bits. ▮

In this paper for simplicity we assume that all messages encrypted have the same length, usually denoted m. This means that $\mu = mq$. We also assume that the length of each of the challenge messages is m. The "time-complexity" refers to the worst case execution time of experiment $\mathbf{Exp}^{\mathrm{priv}}_{\mathsf{SE}}(A)$ plus the size of the code of A, in some fixed RAM model of computation. We are considering only chosen-plaintext attacks, not chosen-ciphertext attacks.

Pseudorandom functions. Consider a map $F\colon \{0,1\}^k \times S \to \{0,1\}^l$ which takes a key $K \in \{0,1\}^k$ and an input x from the domain S to return an output $y = F(K,x)$. The domain S is for convenience $\{0,1\}^*$, or at least the set of all strings of length up to some very large maximum length. The notation $g \stackrel{R}{\leftarrow} F$ is shorthand for $K \stackrel{R}{\leftarrow} \{0,1\}^k$; $g \leftarrow F(K,\cdot)$. We let R denote the family of all functions of S to $\{0,1\}^l$ so that $g \stackrel{R}{\leftarrow} R$ denotes the operation of selecting at random a function of S to $\{0,1\}^l$. A distinguisher D is an algorithm that takes an oracle for a function $g\colon S \to \{0,1\}^l$, and after computing with this oracle returns a bit. The following is the notion of [6] concretized as per [2].

Definition 2.2 Let F, R be as above, let D be a distinguisher, and suppose $t, q, \mu \geq 0$. Define the *advantage of* D, and the *advantage function of* F, respectively, as

$$\begin{aligned} \mathbf{Adv}^{\mathrm{prf}}_{F}(D) &= \Pr\left[\, D^g = 1 \;:\; g \stackrel{R}{\leftarrow} F \,\right] - \Pr\left[\, D^g = 1 \;:\; g \stackrel{R}{\leftarrow} R \,\right] \\ \mathbf{Adv}^{\mathrm{prf}}_{F}(t,q,\mu) &= \max_{D} \{ \, \mathbf{Adv}^{\mathrm{prf}}_{F}(D) \, \} \;. \end{aligned}$$

where the maximum is over all D with time-complexity at most t, making at most q oracle queries, these totalling at most μ bits. ▮

All-or-nothing transforms. Rivest [9] defines an all-or-nothing transform as an efficiently computable, keyless, randomized transformation AONT which maps a message to a sequence of blocks and has the following properties: Given the AONT of some message, one can easily compute the original message, and given all but one of the output blocks, it is computationally infeasible to get any non-trivial information about the message. The (deterministic) inverse transformation permitting recovery of the message from the output is dentoted AONT^{-1}.

We will Rivest's formalization of security —rather than the stronger requirement of Boyko's [5]— because Rivest's notion is easier (or at least, no harder) to achieve and suffices for our new scheme.

We assume for simplicity that the AONT takes input messages of length m and has outputs of length sn. The attack allowed is non-adaptive, meaning the adversary fixes beforehand the position of the output block that will be omitted. Denote this by $L \in \{1,\ldots,s\}$. During the find stage the adversary comes up with a pair of messages M_0 and M_1, both of length m. In its guess stage it is given a AONT for one of the plaintexts M_0, M_1, with block L missing. The adversary wins if it correctly guesses which message goes with the challenge AONT. (The stronger requirement made by Boyko is that an adversary can choose any subset of the bit positions to play the role of the missing bits, not just a block of contiguous bits.) If $X \in \{0,1\}^{sn}$ is a string of s blocks, each n-bits long, then we let $X[1,\ldots,L-1,L+1,\ldots,s]$ denote the string consisting of blocks $1,\ldots,L-1,L+1,\ldots,s$ of X, meaning all but block L.

Definition 2.3 Let AONT be a (possibly randomized) algorithm taking inputs of length m and returning outputs of length sn. Let $L \in \{1,\ldots,s\}$ be a block number. For $b = 0,1$ define the experiment

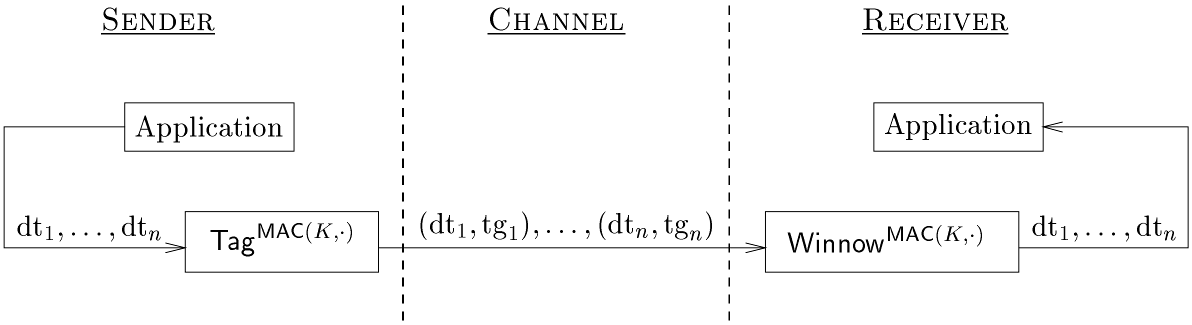


Figure 1: **Authentication channel:** An application on the sender side can put out a data stream $\mathrm{dt}_1, \ldots, \mathrm{dt}_n$. This is passed to a "tagging" procedure which invokes $\mathsf{MAC}(K, \cdot)$ to append a MAC to each item, resulting in a stream of packets $(\mathrm{dt}_1, \mathrm{tg}_1), \ldots, (\mathrm{dt}_n, \mathrm{tg}_n)$ where $\mathrm{tg}_i = \mathsf{MAC}(K, \mathrm{dt}_i)$ for $i = 1, \ldots, n$. This packet stream is transmitted across the channel. A receiving procedure uses $\mathsf{MAC}(K, \cdot)$ to "winnow" an incoming stream of packets: packets with invalid tags are discarded and the data from packets with valid tags is passed up to the receiving application. The applications do not have direct access to $\mathsf{MAC}(K, \cdot)$ let alone to the underlying key K.

Experiment $\mathbf{Exp}^{\mathrm{aont}}_{\mathsf{AONT},L}(A, b)$

$(M_0, M_1, St) \leftarrow A(\mathsf{find})$; $C \leftarrow \mathsf{AONT}(M_b)[1, \ldots, L-1, L+1, \ldots, s]$; $d \leftarrow A(\mathsf{guess}, C, St)$

`Return` d

Now define the *advantage* of A and the *advantage function* of AONT, respectively, as follows:

$$\mathbf{Adv}^{\mathrm{aont}}_{\mathsf{AONT},L}(A) = \Pr\left[\mathbf{Exp}^{\mathrm{aont}}_{\mathsf{AONT},L}(A, 0) = 0\right] - \Pr\left[\mathbf{Exp}^{\mathrm{aont}}_{\mathsf{AONT},L}(A, 1) = 0\right]$$

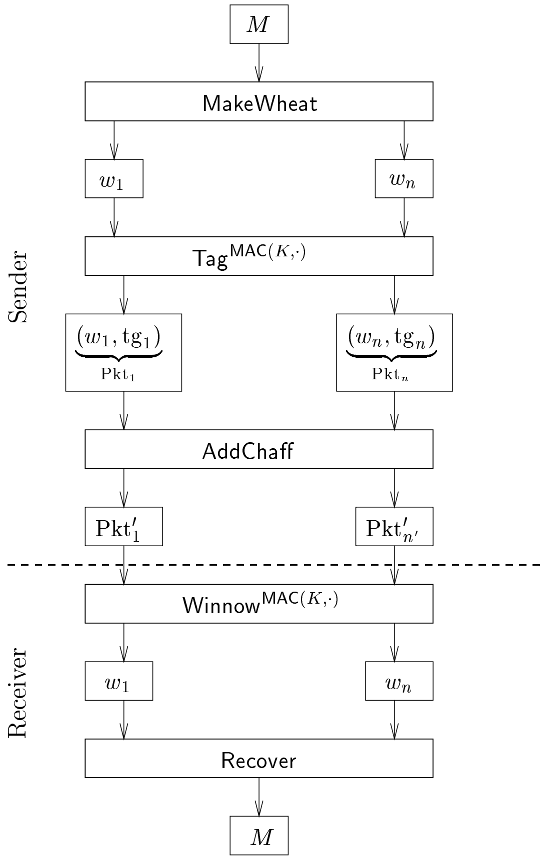
$$\mathbf{Adv}^{\mathrm{aont}}_{\mathsf{AONT},L}(t) = \max_A \{ \mathbf{Adv}^{\mathrm{aont}}_{\mathsf{AONT},L}(A) \}$$

where the maximum is over all A with "time-complexity" t.

3 Defining chaffing and winnowing

Fix for reference a map MAC: $\{0,1\}^k \times \{0,1\}^* \to \{0,1\}^l$ to be used as a message authentication code. (Security assessments will assume that this map is a pseudorandom function, but discussions and constructions will refer to it as a MAC.) A *packet* is a pair $(\mathrm{dt}, \mathrm{tg})$ consisting of *data* dt and a *tag* tg where the length of tg is l-bits, the length of the output of MAC. A packet $(\mathrm{dt}, \mathrm{tg})$ is *valid* with respect to MAC_K —where $K \in \{0,1\}^k$ is some key for the MAC— if $\mathsf{MAC}_K(\mathrm{dt}) = \mathrm{tg}$, and *invalid* with respect to MAC_K otherwise. When MAC_K is understood, we simply talk of valid and invalid packets.

The sender and receiver have an authenticated channel of communication based on the MAC. Each party has a module responsible for authentication. These modules hold in common a key $K \in \{0,1\}^k$ for the MAC. When the sender wants to send data dt to the receiver in an authenticated way, the sender passes dt to its authentication module, which creates the (valid) packet $\mathrm{Pkt} = (\mathrm{dt}, \mathsf{MAC}(K, \mathrm{dt}))$. This packet is sent to the receiver. We call this the "tag" procedure. The packet is received by the receiver's authentication module, which verifies the tag. If the tag is valid, it passes the data "up" to the receiver. If the tag is not valid, the packet is simply discarded; nothing is passed up to the receiver. The receiving module thus acts as a "filter", separating "wheat" (valid) packets from "chaff" (invalid) packets, and passing to the receiver only the data from the valid packets. This is what [8] calls the "winnow" procedure. The two procedures are specified in detail below, and the channel is depicted in Figure 1.



The (plaintext) message M is first processed by a (keyless) transform MakeWheat to yield a sequence $w_1, \ldots, w_n$ of strings, each of which is MACed to yield a stream $\mathrm{Pkt}_1 = (w_1, \mathrm{tg}_1), \ldots, \mathrm{Pkt}_n = (w_n, \mathrm{tg}_n)$ of valid packets, where $\mathrm{tg}_i = \mathsf{MAC}(K, w_i)$ for $i = 1, \ldots, n$. The (keyless) AddChaff procedure adds chaff packets to produce a new stream $\mathrm{Pkt}'_1, \ldots, \mathrm{Pkt}'_{n'}$ of packets (the ciphertext) which is sent to the receiver. They hit the receiver's winnow (cf. Definition 3.1) which discards packets with invalid MACs, and passes up to the receiver the data $\mathrm{dt}_1, \ldots, \mathrm{dt}_n$ from the valid packets. A (keyless) Recover procedure now puts this data together to get back the original message M. The three keyless algorithms MakeWheat, AddChaff, and Recover comprise what we call the ATPT (authentication to privacy transform)—they enable the possibility of obtaining confidentiality via an existing authentication channel without the addition of any extra cryptographic elements.

Figure 2: **Chaff-and-winnow based "encryption".**

Definition 3.1 [MAC-based tag and winnow procedures] We associate to a MAC function $\mathsf{MAC}\colon \{0,1\}^k \times \{0,1\}^* \to \{0,1\}^l$ the following *tag and winnow procedures.* The tag procedure produces a valid packet from the input data. The winnow procedure takes as input a stream of packets and returns the data of the valid packets:

Algorithm $\mathsf{Tag}^{\mathsf{MAC}(K,\cdot)}$	Algorithm $\mathsf{Winnow}^{\mathsf{MAC}(K,\cdot)}(\mathrm{Pkt}'_1, \ldots, \mathrm{Pkt}'_{n'})$
For $i = 1, \ldots, n$ do	For $i = 1, \ldots, n'$ do
$\mathrm{tg}_i \leftarrow \mathsf{MAC}(K, \mathrm{dt}_i)$	Parse Pkt'_i as $(\mathrm{dt}, \mathrm{tg})$
Return $(\mathrm{dt}_i, \mathrm{tg}_i)$	If $\mathsf{MAC}(K, \mathrm{dt}) = \mathrm{tg}$ then return dt
EndFor	EndFor

Here $K \in \{0,1\}^k$ is a key for the MAC and n is the number of packets in the input stream. ▮

The receiver has no direct access to the packets or their MACs, no access to (or even knowledge of) the invalid packets, which are simply discarded by the winnow procedure. The receiver only gets, in order, the data part of the valid packets.

Definition 3.2 [ATPT] An *authentication to privacy transform* (ATPT) with tag length l is a triple $\mathsf{ATPT} = (\mathsf{MakeWheat}, \mathsf{AddChaff}, \mathsf{Recover})$ of algorithms, where

- MakeWheat takes as input a message M and returns a sequence $w_1, \ldots, w_n$ of strings called the *wheat strings*

- AddChaff takes as input a sequence $\mathrm{Pkt}_1, \ldots, \mathrm{Pkt}_n$ of packets called the *wheat packets* and returns another sequence $\mathrm{Pkt}'_1, \ldots, \mathrm{Pkt}'_{n'}$ of packets
- Recover takes as input strings $d_1, \ldots, d_n$ and returns a message M.

The first two algorithms can be probabilistic or stateful (accessing a global state variable such as a counter). The last algorithm is usually deterministic and stateless. ▮

An ATPT above is used in combination with an authentication channel to provide confidentiality. The way the process works is depicted and explained in Figure 2. Our interest is in the security of this entire procedure viewed as a symmetric encryption scheme. For this purpose it is convenient to think of it more as a standard symmetric encryption scheme, consisting of a key generation, encryption and decryption procedure. (The fact that it works by chaff and winnow is irrelevant to the security, although of course crucial to policy debate.) Below, we specify the symmetric encryption scheme that results from running a given ATPT over a given authentication channel, by specifying the three constituent algorithms.

Definition 3.3 Let $\mathsf{ATPT} = (\mathsf{MakeWheat}, \mathsf{AddChaff}, \mathsf{Recover})$ be an ATPT with tag length l, and let $\mathsf{MAC}\colon \{0,1\}^k \times \{0,1\}^* \to \{0,1\}^l$ be a MAC. Associated to them is a *canonical encryption scheme* $(\mathcal{K}, \mathcal{E}, \mathcal{D})$. The key generation algorithm $\mathcal{K}$ is the same as that of the MAC, namely it outputs a random k-bit key K, and the encryption and decryption algorithms are as follows:

```
Algorithm E_K(M)
    (w_1, ..., w_n) ← MakeWheat(M)
    For i = 1, ..., n do
        Pkt_i ← (w_i, MAC_K(w_i))
    EndFor
    (Pkt'_1, ..., Pkt'_{n'}) ← AddChaff(Pkt_1, ..., Pkt_n)
    Return Pkt'_1, ..., Pkt'_{n'}
```

```
Algorithm D_K(Pkt'_1, ..., Pkt'_{n'})
    (dt_1, ..., dt_n) ← Winnow^{MAC(K,·)}(Pkt'_1, ..., Pkt'_{n'})
    M ← Recover(dt_1, ..., dt_n)
    Return M
```

We require that $\mathcal{D}_K(\mathcal{E}_K(M)) = M$ for all $M \in \{0,1\}^*$. ▮

The last requirement is made so that this is a valid symmetric encryption scheme, meaning correctly encrypted data can be decrypted by a receiver that knows the secret key.

In the sequel, we will specify chaff-and-winnow based encryption schemes directly as standard symmetric encryption schemes, because this is more conducive to security assessments. Accordingly it is useful to have the following terminology.

Definition 3.4 Let $\mathsf{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a symmetric encryption scheme. We say that SE is a *chaff-and-winnow based encryption scheme* if there exists an ATPT transform ATPT and a MAC $\mathsf{MAC}\colon \{0,1\}^k \times \{0,1\}^* \to \{0,1\}^l$ such that SE is exactly the canonical confidentiality procedure associated to ATPT and MAC as per Definition 3.3. ▮

Having specified a symmetric encryption scheme, we will briefly indicate why it is a chaff-and-winnow based encryption scheme by saying what are the algorithms MakeWheat, AddChaff, and Recover.

4 Bit-by-Bit CW

As above, $\mathsf{MAC}\colon \{0,1\}^k \times \{0,1\}^* \to \{0,1\}^l$ is a message authentication code. In the bit-by-bit scheme, the sender maintains a counter *ctr* that is initially zero. The encryption procedure (more precisely, the $\mathsf{MakeWheat}$ algorithm) increments this counter upon each invocation. Assume all messages to be encrypted have length m.

Scheme 4.1 [Bit-by-bit CW] The key generation algorithm $\mathcal{K}$ of this symmetric encryption scheme returns a random k-bit key K for the MAC, and the encryption and decryption algorithms are as follows:

Algorithm $\mathcal{E}_K(M)$
  Break M into bits, $M = b_1 \dots b_m$
  For $i = 1, \dots, m$ do
    $\mathrm{tg}[i, b_i] \leftarrow \mathsf{MAC}(K, b_i \| \langle ctr + i \rangle)$
    $\mathrm{tg}[i, \bar{b}_i] \stackrel{R}{\leftarrow} \{0,1\}^l$
    $\mathrm{Pkt}[i, 0] \leftarrow (0 \| \langle ctr + i \rangle, \mathrm{tg}[i, 0])$
    $\mathrm{Pkt}[i, 1] \leftarrow (1 \| \langle ctr + i \rangle, \mathrm{tg}[i, 1])$
  EndFor
  $ctr \leftarrow ctr + m$
  Return $\mathrm{Pkt}[1,0], \mathrm{Pkt}[1,1], \dots, \mathrm{Pkt}[m,0], \mathrm{Pkt}[m,1]$
	Algorithm $\mathcal{D}_K(\mathrm{Pkt}_1, \dots, \mathrm{Pkt}_{2m})$
	  For $i = 1, \dots, 2m$ do
	    Parse Pkt_i as $(\mathrm{dt}, \mathrm{tg})$
	    If $\mathsf{MAC}(K, \mathrm{dt}) = \mathrm{tg}$
	    then return first bit of dt
	  EndFor

Here $\bar{b}$ denotes the complement bit of b and $\langle i \rangle$ denotes the binary representation of integer i as a binary string of some fixed, predefined length p. The "wheat" packets are $\mathrm{Pkt}[i, b_i]$ for $i = 1, \dots, m$ and the "chaff" packets are $\mathrm{Pkt}[i, \bar{b}_i]$ for $i = 1, \dots, m$.

We claim that this is a chaff-and-winnow based encryption scheme. We need to check that it has the form of Definition 3.3 for some ATPT. The $\mathsf{MakeWheat}$ procedure takes M and returns $b_1 \| \langle ctr \rangle, \dots, b_n \| \langle ctr + m \rangle$. The $\mathsf{AddChaff}$ procedure would get as input the wheat packets corresponding to this data stream, namely $\mathrm{Pkt}[1, b_1], \dots, \mathrm{Pkt}[m, b_m]$. It would create the packets $\mathrm{Pkt}[1, \bar{b}_1], \dots, \mathrm{Pkt}[m, \bar{b}_m]$ and then output them in the order indicated by the output of $\mathcal{E}_K$ above. The $\mathsf{Recover}$ procedure would receive $b_1 \| \langle ctr \rangle, \dots, b_m \| \langle ctr + n \rangle$ and would drop the counter values to return $b_1 \dots b_m$. ▮

The following theorem shows that this scheme meets the "find-then-guess" notion of privacy under the assumption that MAC is a PRF. The reduction is almost tight.

Theorem 4.2 Let MAC: $\{0,1\}^k \times \{0,1\}^* \to \{0,1\}^l$ be a pseudorandom function and let $\mathsf{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be the bit-by-bit chaff-and-winnow based encryption scheme of Scheme 4.1. Assume the counter is p-bits long. Then for any t, q, μ with $\mu < 2^{p}-$

$$\mathbf{Adv}^{\mathrm{priv}}_{\mathsf{SE}}(t, q, \mu) \leq 2 \cdot \mathbf{Adv}^{\mathrm{prf}}_{\mathsf{MAC}}(t, q', \mu') ,$$

where $q' = \mu$ and $\mu' = (1 + p)\mu$.

Proof of Theorem 4.2: Let A be any adversary trying to break the bit-by-bit CW encryption scheme in the find-then-guess setting as described in Section 2. We will design a distinguisher D which attacks the MAC as a pseudorandom function. D takes an oracle g: $\{0,1\}^* \to \{0,1\}^l$. The distinguisher does not know a priori whether g is drawn at random from F or from R. To figure it out D will use A. D will provide the ciphertext of one of the two messages found by A during its find stage and will respond to all oracle queries that A makes. Assume A makes q oracle queries totaling μ bits, and runs for the time t. Here is what D does:

Distinguisher $D^{g(\cdot)}$
  $b \stackrel{R}{\leftarrow} \{0,1\}$; $(M_0, M_1, st) \leftarrow A^{\mathcal{E}^g}(\mathsf{find})$; $C \leftarrow \mathcal{E}^g(M_b)$; $d \leftarrow A^{\mathcal{E}^g}(\mathsf{guess}, C, st)$
  If $b = d$ then return 1 (PRF) else return 0 (random)

Here $\mathcal{E}^g(\cdot)$ denotes the procedure which substitutes all applications of $\mathsf{MAC}(K,\cdot)$ in $\mathcal{E}_K$ with an application of $g(\cdot)$. It is possible for D to compute this as above because $\mathcal{E}_K$ only makes oracle use of $\mathsf{MAC}(K,\cdot)$. We now analyze this distinguisher. We claim that

$$\Pr\left[\, D^g = 1 \;:\; g \stackrel{R}{\leftarrow} F \,\right] = \Pr\left[\, b = d \;:\; g \stackrel{R}{\leftarrow} F \,\right] = \frac{1}{2} + \frac{\mathbf{Adv}^{\mathrm{priv}}_{\mathsf{SE}}(A)}{2}$$

$$\Pr\left[\, D^g = 1 \;:\; g \stackrel{R}{\leftarrow} R \,\right] = \Pr\left[\, b = d \;:\; g \stackrel{R}{\leftarrow} R \,\right] = \frac{1}{2} + \frac{\mathbf{Adv}^{\mathrm{priv}}_{\mathsf{SE}^R}(A)}{2}$$

where SE^R denotes the encryption scheme using a random function in place of the MAC. Because of the globally incremented counter, oracle g is always invoked on distinct inputs. If oracle g is a random function, then its output on these distinct values yields a one-time pad, meaning random and unpredictable values, so the adversary cannot get any advantage. This means that $\mathbf{Adv}^{\mathrm{priv}}_{\mathsf{SE}^R}(A) = 0$. Now by subtraction we get $\mathbf{Adv}^{\mathrm{priv}}_{\mathsf{SE}^F}(A) \leq 2 \cdot \mathbf{Adv}^{\mathrm{prf}}_{F}(D)$. The statement of the theorem follows after taking maximums. ∎

If the counter is allowed to wrap around the scheme is obviously insecure. It is possible to use randomness instead of a counter. In this case each bit of the message is concatenated with random value represented as a string of some fixed predefined length. This value is drawn at random for each bit of the message. The analysis is analogous but the concrete security is worse due to birthday attacks.

The security of these schemes comes with a price. They are very inefficient since they have large data expansion: if the message is m bits long then $2m(1+p+l)$ bits are transmitted, where p is the length of a counter and l is the length of the output of the MAC.

Another scheme mentioned in [8] authenticated message blocks of some length rather than single bits, but the author already indicated that it was insecure under stringent notions of privacy such as the one we use here.

5 The scattering scheme

As before let $\mathsf{MAC}\colon \{0,1\}^k \times \{0,1\}^* \to \{0,1\}^l$ be our message authentication code. In the scattering scheme, the output of an AONT is viewed as a sequence of s blocks, each n bits long. The ciphertext will contain s wheat packets interspaced with $s' > 0$ chaff packets, where s' is a parameter of the scheme. The wheat packet positions are a random subset of $\{1, \ldots, s+s'\}$. The full description follows.

Scheme 5.1 [Scattering scheme] We fix an all-or-nothing transform $\mathsf{AONT}\colon \{0,1\}^m \to \{0,1\}^{sn}$. We assume that all messages to be encrypted have length m. The key generation algorithm $\mathcal{K}$ of this symmetric encryption scheme returns a random k-bit key K for the MAC, and the encryption and decryption algorithms are as follows:

```
Algorithm E_K(M)
    M' ← AONT(M)
    Parse M' as m_1||m_2||···||m_s where |m_i| = n
    Pick S ⊆ {1,...,s+s'} at random subject to |S| = s
    j ← 0
    For i = 1,...,s+s' do
        If i ∈ S then
            j ← j+1
            tg[i] ← MAC(K, m_j)
            Pkt[i] ← (m_j, tg[i])
        else
            dt[i] ←R {0,1}^n
            tg[i] ←R {0,1}^l
            Pkt[i] ← (dt[i], tg[i])
        EndIf
    EndFor
    Return Pkt[1], Pkt[2], ... Pkt[s+s']
```

```
Algorithm D_K(Pkt_1,...,Pkt_{s+s'})
    For i = 1,...,s+s' do
        Parse Pkt_i as (dt, tg)
        If MAC(K, dt) = tg
        then m_i ← dt
    EndFor
    M ← AONT^{-1}(m_1||m_2||···||m_s)
    Return M
```

We claim that this is a chaff-and-winnow based encryption scheme. We need to check that it has the form of Definition 3.3 for some ATPT. The MakeWheat procedure takes M and returns $m_1, \ldots, m_s$ as computed above. The AddChaff procedure would get as input the wheat packets corresponding to this data stream. It would create the s' chaff packets, pick the set S at random as above, and interspace the wheat and chaff packets appropriately to get the packet stream $\mathrm{Pkt}[1], \ldots, \mathrm{Pkt}[s+s']$. The Recover procedure would receive $m_1, \ldots, m_s$, concatenate them to get M', and apply AONT^{-1} to get M. ▮

The most obvious attack is to test each group of s packets to see whether they are the wheat packets. The adversary goes through all size s subsets of the packets. In each case it forms a candidate output of AONT and applies AONT^{-1}. Assuming it knows some partial information about the message, it can tell when it got the choice of the subset right. The time taken by this attack is proportional to $\binom{s+s'}{s}$.

The intuition for security given in [8] is that this is the best possible attack. The complexity is large as long as both s and s' are above some minimal threshold; for example, both more than 128. Accordingly we could set $s' = 128$ and choose the AONT so that its output always had at least 128 blocks.

A closer look reveals however that security is not so straightforward. For example, another thing to consider is the effect of equal data blocks. If the data blocks in two packets are equal, an adversary can get some information by looking at their tags: if the tags are unequal, they cannot both be wheat packets, because the MAC is deterministic. This can reduce the complexity of an attack, indicating that the time-complexity of an attack must also be a function of the block size n of the output.

There are other such considerations, but more importantly, we claim that Rivest's notion of security for the AONT (namely Definition 2.3) can be shown to be insufficient to make this scheme secure. An example illustrating this is to consider an AONT each of whose output blocks has the property that the first few bits are 0. (One can show that if any AONT meeting Definition 2.3 exists, then so does one with this property.) But with this AONT, Scheme 5.1 can be broken because wheat packets can be distinguished from chaff packets: the wheat packets are the ones whose data has first few bits zero.

The example AONT we constructed above does not however meet Boyko's stronger notion of security for AONTs [5], so the next question is whether Scheme 5.1 could be proven secure under this stronger notion. However even with this stronger notion it is unclear one can prove security. The reason is that the ciphertext contains the complete output of the AONT, while the security property of the AONT pertains to a setting where the adversary has no information about at least one block of the output of the AONT. This makes it unclear how to do a reduction. Indeed, the security property of an AONT does not seem to mesh well with what is required to prove security of Scheme 5.1. We will see next that a positive statement can be made by considering a particular AONT, namely the OAEP transform of Bellare and Rogaway [4]. But in general, as the transform used in the initial step, an AONT seems to be neither sufficient nor necessary for the security of Scheme 5.1.

The OAEP transform appeals to random oracles $G\colon \{0,1\}^n \to \{0,1\}^m$ and $H\colon \{0,1\}^m \to \{0,1\}^n$ where n is the length of the OAEP seed and m as usual is the message length. It takes as input an m-bit string M and proceeds as follows–

Algorithm $\mathsf{OAEP}^{G,H}(M)$
$r \stackrel{R}{\leftarrow} \{0,1\}^n$; $y \leftarrow G(r) \oplus M$; $w \leftarrow H(y) \oplus r$; **Return** $w \| y$

Boyko showed that OAEP is an AONT, but this will not help us here given the above discussion. Instead, we go back to the transform itself and prove the security of Scheme 5.1 when AONT is set to OAEP.

As with any proof concerning OAEP, we work in the random oracle model of [3]. We must "lift" our definitions to allow all algorithms and parties, including the adversary, access to the random oracles G, H. Briefly, modify $\mathbf{Exp}^{\mathrm{priv}}_{\mathsf{SE}}(A,b)$ in Definition 2.1 to begin by picking G, H randomly. Allow $\mathcal{E}_K$ and A oracle access to G, H. Allow the scheme advantage to take extra parameters, $\mathbf{Adv}^{\mathrm{priv}}_{\mathsf{SE}}(t,q,\mu;q_G,q_H)$, these being bounds on the number of queries made by the adversary to the oracles in question. T

The bound below reflects the above intuition: it is inversely proportional to $\binom{s+s'}{s}$ and also to 2^n. This shows that for OAEP the security is what one would have liked it to be for a "good" AONT.

Theorem 5.2 Let n, m be integers with m a multiple of n. Let $\mathsf{MAC}\colon \{0,1\}^k \times \{0,1\}^* \to \{0,1\}^l$ be a pseudorandom function. Let $\mathsf{SE} = (\mathcal{K},\mathcal{E},\mathcal{D})$ be Scheme 5.1 using OAEP as the AONT, with parameters n, s, s' where $s = m/n+1$. For any t, q we let $\mu = nq(s-1)$. Assume $q_H \le 2^n/2$ and $q_G \le (1/2)\cdot\binom{s+s'}{s}$. Then

$$\mathbf{Adv}^{\mathrm{priv}}_{\mathsf{SE}}(t,q,\mu;q_G,q_H) \;\le\; \frac{2q_H}{\binom{s+s'}{s-1}} + \frac{2q_G + (q+1)^2\cdot[(s+s')^2+1]}{2^n} + \mathbf{Adv}^{\mathrm{prf}}_{\mathsf{MAC}}(t',q',\mu')$$

where $t' = t$, $q' = q(s+s')$ and $\mu' = nq(s+s')$.

Proof of Theorem 5.2: The main step in the analysis is to consider the scheme which uses, in place of $\mathsf{MAC}(K,\cdot)$, a truly random function of the same domain and range. Now we consider an adversary making q queries to its encryption oracle, q_G queries to G, and q_H queries to H, and show that in this case

$$\mathbf{Adv}^{\mathrm{priv}}_{\mathsf{SE}}(A) \;\le\; \frac{2q_H}{\binom{s+s'}{s-1}} + \frac{2q_G + (q+1)^2\cdot[(s+s')^2+1]}{2^n}\,. \tag{1}$$

Once we have shown this, the theorem follows by a standard simulation argument for pseudorandom functions which is omitted.

In its find stage A outputs challenge messages M_0, M_1. Then it gets a challenge ciphertext C which is an encryption of M_b for a challenge bit b. Let r, y denote the values in the computation of $\mathsf{OAEP}^{G,H}(M_b)$. Let AskR be the event that A makes G-oracle query r. Let AskY be the event that A makes H-oracle query y. Let Corr denote the event that A is correct, meaning its output equals the bit b. Then

$$\begin{aligned}
\Pr[\,\mathsf{Corr}\,] &= \Pr[\,\mathsf{Corr} \mid \mathsf{AskR}\,] \cdot \Pr[\,\mathsf{AskR}\,] + \Pr[\,C \mid \neg\mathsf{AskR}\,] \cdot \Pr[\,\neg\mathsf{AskR}\,] \\
&\leq \Pr[\,\mathsf{AskR}\,] + \Pr[\,C \mid \neg\mathsf{AskR}\,] \\
&\leq \Pr[\,\mathsf{AskR}\,] + \frac{1}{2}\,.
\end{aligned}$$

The last term is because if A did not see $G(r)$, it has no advantage in predicting b. So we need to upper bound $\Pr[\mathsf{AskR}]$. Let AskR^i and AskY^i denote the events that A's i'th query to a random oracle is r or y respectively. Then

$$\begin{aligned}
\Pr[\,\mathsf{AskR}\,] &\leq \Pr[\,\mathsf{AskR} \vee \mathsf{AskY}\,] \\
&\leq \Pr\left[\,\mathsf{AskR}^1 \vee \mathsf{AskY}^1\,\right] + \sum_{i=2}^{q_G+q_H} \Pr\left[\,(\mathsf{AskR}^i \vee \mathsf{AskY}^i) \mid \bigwedge_{j=1}^{i-1} \neg\mathsf{AskR}^j \wedge \neg\mathsf{AskY}^j\,\right]\,.
\end{aligned}$$

As long as r was a query to G and y was not a query to H the adversary knows nothing about $H(y)$. So $r = H(y) \oplus w$ could be any value. By the same reasoning an adversary does not know anything about the value of $y = G(r) \oplus M$. Let Dist denote the event that all the $(q+1)(s+s')$ blocks that form the output of the OAEP transform on the queried messages and on the challenge messages are distinct. (If $\neg$Dist then an adversary can get some information by looking at the tags of equal blocks as discussed above.) We will say the adversary has won if Dist occurs. Now we claim that for any $i \geq 1$ we have

$$\begin{aligned}
\Pr\left[\,\mathsf{AskR}^i \mid \mathsf{Dist} \wedge \textstyle\bigwedge_{j=1}^{i-1} \neg\mathsf{AskR}^j \wedge \neg\mathsf{AskY}^j\,\right] &\leq \frac{1}{2^n - (i-1)} \\
\Pr\left[\,\mathsf{AskY}^i \mid \mathsf{Dist} \wedge \textstyle\bigwedge_{j=1}^{i-1} \neg\mathsf{AskR}^j \wedge \neg\mathsf{AskY}^j\,\right] &\leq \frac{1}{\binom{s+s'}{s-1} - (i-1)}\,.
\end{aligned}$$

This is by the above reasoning combined with the fact that if Dist occurs then there are $\binom{s+s'}{s-1}$ *different* possible choices for y. (Recall y is $s-1$ blocks long.) We assumed that $q_H \leq 2^n/2$ and $q_G \leq (1/2) \cdot \binom{s+s'}{s}$. So

$$\Pr[\,\mathsf{AskR} \mid \mathsf{Dist}\,] \leq \sum_{j=0}^{q_G-1} \frac{1}{2^n - j} + \sum_{j=0}^{q_H-1} \frac{1}{\binom{s+s'}{s-1} - j} \leq \frac{2q_G}{2^n} + \frac{2q_H}{\binom{s+s'}{s-1}}\,.$$

On the other hand Dist can fail for two reasons. One is that the same seed was picked twice. Another is a standard birthday collision. Together this means that $\Pr[\,\neg\mathsf{Dist}\,] \leq (q+1)^2 \cdot [(s+s')^2 + 1]/2^n$. Putting all this together gives us Equation (1). ▮

6 A new chaffing and winnowing scheme

Here we suggest an alternative scheme that has low data expansion and analyze its advantage function. It returns to much more "standard" paradigms of encryption than the scattering scheme. Simply apply an AONT to the message and then encrypt the first block of the message. If the last encryption is done by chaffing-and-winnowing, say using the bit-by-bit scheme, the whole scheme is also a chaffing and winnowing scheme, since the AONT is keyless. The savings in bandwidth comes from the fact that the number of bits encrypted using the bit-by-bit scheme is independent of the length of the message. The construction is presented using an arbitrary symmetric encryption scheme se to encrypt the first block of the output of the AONT, since the analysis can be done at this general level.

Scheme 6.1 Let AONT be an all-or-nothing transform taking input messages of length m and returning outputs of length sn. The output is viewed as a sequence of n-bit blocks. Let $\mathsf{se} = (\mathcal{K}, \mathsf{e}, \mathsf{d})$ be a given symmetric encryption scheme with message space $\{0,1\}^n$. The new scheme is $\mathsf{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ where

Algorithm $\mathcal{E}_K(M)$
    $M' \leftarrow \mathsf{AONT}(M)$
    Let m' be the first block of M' and m'' the rest
    $C_1 \leftarrow \mathsf{e}_K(m')$
    Return $C_1 \| (m'', \mathsf{MAC}(K, m''))$

Algorithm $\mathcal{D}_K(C_1 \| (m'', \tau))$
    $m' \leftarrow \mathsf{d}_K(C_1)$
    $M' \leftarrow m' \| m''$
    $M \leftarrow \mathsf{AONT}^{-1}(M')$
    Return M

We claim that if se is Scheme 4.1 then the above is a chaff-and-winnow based encryption scheme. We need to check that it has the form of Definition 3.3 for some ATPT. Note that when se is Scheme 4.1, C_1 will have the form $\mathrm{Pkt}[1,0], \mathrm{Pkt}[1,1], \ldots, \mathrm{Pkt}[n,0], \mathrm{Pkt}[n,1]$. The $\mathsf{MakeWheat}$ procedure takes M and returns $b_1 \| \langle ctr \rangle, \ldots, b_n \| \langle ctr + n \rangle, m''$, where $b_1 \ldots b_n$ are the first n bits of $\mathsf{AONT}(M)$ and the m'' is the rest. The $\mathsf{AddChaff}$ procedure would get as input the wheat packets corresponding to this data stream, namely $\mathrm{Pkt}[1, b_1], \ldots, \mathrm{Pkt}[n, b_n], (m'', \mathsf{MAC}(K, m''))$. It would create the packets $\mathrm{Pkt}[1, \bar{b}_1], \ldots, \mathrm{Pkt}[n, \bar{b}_n]$ and then output

$$\mathrm{Pkt}[1,0], \mathrm{Pkt}[1,1], \ldots, \mathrm{Pkt}[n,0], \mathrm{Pkt}[n,1], (m'', \mathsf{MAC}(K, m'')) \ .$$

The $\mathsf{Recover}$ procedure would receive $b_1 \| \langle ctr \rangle, \ldots, b_n \| \langle ctr + n \rangle, m''$ and would drop the counter values to get $M' = b_1 \ldots b_n \| m''$, perform $\mathsf{AONT}^{-1}(M')$ and return M. ∎

More generally, the above is a chaff-and-winnow based encryption scheme as long as se is any chaff-and-winnow based encryption scheme, not just Scheme 4.1.

We now analyze the security of Scheme 6.1 at a general level. Refer to Definition 2.3 for the definition of the advantage function of AONT and note that $L = 1$ in this case, meaning we are requiring security only in the case where the first block is the one not provided to the adversary.

Note that the MAC attached to m'' is irrelevant to security; it is only there in order to make the final scheme a chaffing and winnowing scheme. Accordingly, the advantage function of the MAC does not show up in the following theorem.

Theorem 6.2 Let $\mathsf{se} = (\mathcal{K}, \mathsf{e}, \mathsf{d})$ be a given, secure symmetric encryption scheme with message space $\{0,1\}^n$. Let AONT be a given, secure all-or-nothing transform. We associate to them the symmetric encryption scheme $\mathsf{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ of Scheme 6.1, with message space $\{0,1\}^m$. Let $\mu = qm$. Then

$$\mathbf{Adv}^{\mathrm{priv}}_{\mathsf{SE}}(t, q, \mu) \quad \leq \quad 2 \cdot \mathbf{Adv}^{\mathrm{priv}}_{\mathsf{se}}(t, q, qn) + \mathbf{Adv}^{\mathrm{aont}}_{\mathsf{AONT},1}(t) \ .$$

This is proved below. First however we state the corollary for the case where se is the bit-by-bit chaff-and-winnow scheme.

Corollary 6.3 Let MAC: $\{0,1\}^k \times \{0,1\}^* \to \{0,1\}^l$ be a pseudorandom function and let AONT be an all-or-nothing transform with input length m, output length sn. Let $\mathsf{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be Scheme 6.1 using AONT as the all-or-nothing transform and Scheme 4.1 as se. Assume the counter in the latter is p-bits long. Then for any t, q, μ with $\mu = qm$ and $qn < 2^{p}$–

$$\mathbf{Adv}^{\mathrm{priv}}_{\mathsf{SE}}(t,q,\mu) \quad \leq \quad 4 \cdot \mathbf{Adv}^{\mathrm{prf}}_{\mathsf{MAC}}(t,q_1,\mu_1) + \mathbf{Adv}^{\mathrm{aont}}_{\mathsf{AONT},1}(t)$$

where $q_1 = qn$ and $\mu_1 = (s+p)n$.

Proof: This follows from Theorem 6.2 and Theorem 4.2. Details omitted. ▌

Proof of Theorem 6.2: Let A be an adversary attacking the scheme SE. We will construct an adversary B attacking se and an adversary C attacking AONT such that

$$\mathbf{Adv}^{\mathrm{priv}}_{\mathsf{SE}}(A) \;=\; 2 \cdot \mathbf{Adv}^{\mathrm{priv}}_{\mathsf{se}}(B) + \mathbf{Adv}^{\mathrm{aont}}_{\mathsf{AONT},1}(C)\ . \tag{2}$$

Furthermore the resources used by B, C are related to those of A in the appropriate manner to yield the theorem. In the following we will for simplicity ignore the MAC function since it won't affect security.

We first define four (hybrid) experiments. Let $\mathsf{e}_K(\cdot), \mathcal{E}_K(\cdot)$ denote the encryption procedures of se, SE respectively.

Experiment 1
- $K \xleftarrow{R} \{0,1\}^k$
- $(M_0, M_1, St) \leftarrow A^{\mathcal{E}_K(\cdot)}(\mathsf{find})$
- $A_0 \leftarrow \mathsf{AONT}(M_0)$
- Let A_0' be the first block of A_0 and A_0'' the rest
- $m \xleftarrow{R} \{0,1\}^r$
- $C \leftarrow \mathsf{e}_K(m) || A_0''$
- $c \leftarrow A^{\mathcal{E}_K(\cdot)}(\mathsf{guess}, C, St)$
- `Return` c

Experiment 2
- $K \xleftarrow{R} \{0,1\}^k$
- $(M_0, M_1, St) \leftarrow A^{\mathcal{E}_K(\cdot)}(\mathsf{find})$
- $A_1 \leftarrow \mathsf{AONT}(M_1)$
- Let A_1' be the first block of A_1 and A_1'' the rest
- $m \xleftarrow{R} \{0,1\}^r$
- $C \leftarrow \mathsf{e}_K(m) || A_1''$
- $c \leftarrow A^{\mathcal{E}_K(\cdot)}(\mathsf{guess}, C, St)$
- `Return` c

Experiment 3
- $K \xleftarrow{R} \{0,1\}^k$
- $(M_0, M_1, St) \leftarrow A^{\mathcal{E}_K(\cdot)}(\mathsf{find})$
- $A_0 \leftarrow \mathsf{AONT}(M_0)$
- Let A_0' be the first block of A_0 and A_0'' the rest
- $C \leftarrow \mathsf{e}_K(A_0') || A_0''$
- $c \leftarrow A^{\mathcal{E}_K(\cdot)}(\mathsf{guess}, C, St)$
- `Return` c

Experiment 4
- $K \xleftarrow{R} \{0,1\}^k$
- $(M_0, M_1, St) \leftarrow A^{\mathcal{E}_K(\cdot)}(\mathsf{find})$
- $A_1 \leftarrow \mathsf{AONT}(M_1)$
- Let A_1' be the first block of A_1 and A_1'' the rest
- $C \leftarrow \mathsf{e}_K(A_1') || A_1''$
- $c \leftarrow A^{\mathcal{E}_K(\cdot)}(\mathsf{guess}, C, St)$
- `Return` c

Let P_i be the probability that experiment i outputs 0. By definition of the advantage of A

$$\mathbf{Adv}^{\mathrm{priv}}_{\mathsf{SE}}(A) \;=\; P_3 - P_4 \;=\; (P_3 - P_1) + (P_2 - P_4) + (P_1 - P_2)\ . \tag{3}$$

Now we define the algorithms B and C as follows

```
Adversary B^{e_K(·)}(find)
    b ←R {0,1}
    (M_0, M_1, St) ← A^{𝓔_K(·)}(find)
    m ←R {0,1}^r
    A_b ← AONT(M_b)
    Let A'_b be the first r bits of A_b and A''_b the rest
    If b = 0 then
        return(A'_0, m, (A''_0, St))
    else
        return(m, A'_1, (A''_1, St))
    EndIf

Adversary B^{e_K(·)}(guess, C, (A''_b, St))
    C ← C||A''_b
    d ← A^{𝓔_K(·)}(guess, C, St)
    Return d
```

```
Adversary C(find)
    (M_0, M_1, St) ← A^{𝓔_K(·)}(find)
    Return (M_0, M_1, St)

Adversary C(guess, C, St)
    K ←R {0,1}^k
    m ←R {0,1}^r
    C ← e_K(m)||C
    d ← A^{𝓔_K(·)}(guess, C, St)
    Return d
```

Both adversaries B and C can simulate the encryption oracle $\mathcal{E}_K(\cdot)$ for A. B does it using its own encryption oracle $\mathsf{e}_K(\cdot)$ and computing the AONT of a desired message, while C computes the AONT and encrypts the first part of a desired message by choosing the random key and simulating the e_K procedure. The advantage of B is

$$\begin{aligned}\mathbf{Adv}^{\mathrm{priv}}_{\mathsf{se}}(B) &= \Pr\left[\,\mathbf{Exp}_{\mathsf{SE}}(B,0)=0\,\right] - \Pr\left[\,\mathbf{Exp}_{\mathsf{SE}}(B,1)=0\,\right] \\ &= \frac{1}{2}\cdot(P_3+P_2) - \frac{1}{2}\cdot(P_4+P_1) \\ &= \frac{1}{2}\cdot(P_3-P_1) + \frac{1}{2}\cdot(P_2-P_4)\ .\end{aligned}$$

The advantage of C is

$$\begin{aligned}\mathbf{Adv}^{\mathrm{aont}}_{\mathsf{AONT},1}(C) &= \Pr\Big[\,\mathbf{Exp}_{\mathsf{AONT},r}(C,0)=0\,\Big] - \Pr\Big[\,\mathbf{Exp}_{\mathsf{AONT},r}(C,1)=0\,\Big] \\ &= P_1 - P_2\ .\end{aligned}$$

Combining these two observations with Equation (3) yields Equation (2). The rest of the proof is standard and is omitted. ▮

A concrete instantiation can be obtained by using OAEP in the role of the AONT. The security of this instantiation relies on the fact that OAEP is a secure AONT [5], and the concrete security can be obtained by combining the above with the results in [5]. (In that case we would have to lift all of the above to the random oracle model, but this is easily done.)

Acknowledgments

References

[1] M. Bellare, A. Desai, E. Jokipii and P. Rogaway, "A concrete security treatment of symmetric encryption: Analysis of the DES modes of operation," *Proceedings of the* 38th *Symposium on Foundations of Computer Science*, IEEE, 1997.

[2] M. Bellare, J. Kilian and P. Rogaway, "The security of cipher block chaining," *Advances in Cryptology – Crypto* 94 *Proceedings*, Lecture Notes in Computer Science Vol. 839, Y. Desmedt ed., Springer-Verlag, 1994.

[3] M. Bellare and P. Rogaway, "Random oracles are practical: a paradigm for designing efficient protocols," *Proceedings of the 1st Annual Conference on Computer and Communications Security*, ACM, 1993.

[4] M. Bellare, P. Rogaway, "Optimal asymmetric encryption - How to encrypt with RSA," *Advances in Cryptology – Eurocrypt* 94 *Proceedings*, Lecture Notes in Computer Science Vol. 950, A. De Santis ed., Springer-Verlag, 1994.

[5] V. Boyko, "On the Security Properties of OAEP as an All-or-nothing Transform,"*Advances in Cryptology – Crypto* 99 *Proceedings*, Lecture Notes in Computer Science Vol. 1666, M. Wiener ed., Springer-Verlag, 1999.

[6] O. Goldreich, S. Goldwasser and S. Micali, "How to construct random functions,"*Journal of the ACM,* Vol. 33, No. 4, 210–217, (1986).

[7] S. Goldwasser and S. Micali, "Probabilistic encryption," *Journal of Computer and System Science*, Vol. 28, 1984, pp. 270–299.

[8] R. Rivest, "Chaffing and Winnowing: Confidentiality without Encryption," `http://theory.lcs.mit.edu/~rivest/publications.html`.

[9] R. Rivest, "All-Or-Nothing Encryption and The Package Transform," *Proceedings of the 4th Workshop on Fast Software Encryption*, Lecture Notes in Computer Science Vol. 1267, Springer-Verlag, 1997.