SECURITY ANALYSIS OF BBC CODING

LEEMON C. BAIRD III AND WILLIAM L. BAHN

Abstract. In large-scale, wireless networks, it is possible to achieve confidentiality and integrity using asymmetric keys, but availability (i.e. resistance to jamming) has always required symmetric keys. Key management for these symmetric keys is becoming increasingly difficult, and in some cases physically impossible. Recently, the first algorithm was proposed to achieve availability without symmetric keys: BBC coding, which is an unkeyed coding system built up from a hash function. However, no security analysis of it has ever been published. In this paper, we prove BBC has unconditional security with high probability in the random oracle model. Furthermore, we prove BBC has unconditional security with high probability for particular families of polynomial-time and -space hash functions. Unlike the random oracle proof, the latter proof applies to systems that can actually be implemented in practice. In addition, we prove that certain families of hash functions that might naturally be considered are actually insecure, and we give methods for breaking them. Finally, we describe the open question of whether BBC is secure for certain families of simple hash functions.

1. Introduction

Key management became much easier in the 1970s when the first asymmetric encryption and digital signature systems became publicly known. Until recently, no equivalent advance had been made for jam-resistant communication in omnidirectional radios. All known systems for that problem required a symmetric key, which was then used in an algorithm equivalent to steganography.

For example, in a frequency hopping system, Alice uses a symmetric key to generate a pseudorandom sequence of numbers which are interpreted as frequencies. She then broadcasts a few bits of her message on each frequency before jumping to the next. Bob uses the same key to generate the same sequence, and listens on each frequency in turn. If a jammer James does not know the secret key, then he must broadcast noise on all of the frequencies (or many of them) continuously, and so must expend much more energy to jam the signal than Alice used to send it. So the signal is jam resistant. However, if James knows the key, he can broadcast on just the sequence of frequencies generated by the key, using about the same amount of energy as Alice, and so the system is no longer jam resistant.

Other forms of spread spectrum radio have also been proposed for jam resistance (e.g. direct sequence or pulse-based ultra wideband), but these, too, rely on a shared secret key and a form of steganography to resist jamming. For example, in a system based on pulses, the message is encoded as a sequence of very short, very powerful pulses of radio noise, each of which spans a large portion of the available spectrum. The information is conveyed in the timing of the pulses. In one example, a 1000-bit message is sent during 1 second, which is divided into a billion time slices. Alice uses her symmetric key to pseudorandomly choose 2000 of those billion time slices. Each bit of her message is sent using 2 of those times. If her bit is a 0, she sends a pulse at the first of the two times. If her bit is a 1, she sends it at the second. Again, if James does not know the key, then he can only jam by sending a pulse at almost every point in time, thus using at least thousands of times more energy than Alice. But if James knows the key, then he can send just 2000 pulses, filling in all the selected time slots, and effectively preventing any information from being received by Bob.

The use of symmetric keys makes key management problematic. For example, handheld jammers for cell phones can now be bought on the open market for under $200 [1]. The cell phone carriers do not build in any resistance to jamming at all. If a carrier wanted to resist jamming, it would have to give each of its millions of subscribers a different secret key. Then, each cell phone tower would have simultaneously listen

This work was sponsored in part by the Air Force Information Operations Center (AFIOC), Lackland AFB, TX, and was performed at the Academy Center for Cyberspace Resaerch (ACCR) at the United States Air Force Academy.

on all of the millions of channels, continuously at all times, to establish a jam-resistant connections with any possible user who might enter the cell. That is impractical. Symmetric key management systems such as Kerberos, which can distribute session keys as needed, are of no help in this case, because they assume each user can communicate with a central server without being jammed.

An even worse case is the civilian Global Positioning System (GPS). In the US, the FAA has stated that its intention is for the GPS system to be used as a primary navigation aid on all aircraft, and that every aircraft will continuously broadcast its GPS position to all other aircraft in the area, thus allowing aircraft to be packed more densely in the air over crowded airports. If a terrorist were to jam the signal, this could be a problem. Yet the system has no form of jam resistance at all. That is because the only known systems were based on symmetric keys. That would require one copy of the key to be on the satellite, and another copy to be given to every legitimate user of the system. But the user base for GPS includes every person on the planet, including the terrorists. So it is literally impossible to prevent the jammer from obtaining the symmetric key. Asymmetric keys would be easy to manage here: just put the private key on the satellite and distribute the corresponding public key throughout the world. However, no form of jam-resistant coding was known that could use asymmetric keys. Only symmetric keys could be used.

The BBC algorithm was recently proposed to solve this problem [2, 3]. It is an encoding system that requires no keys at all for jam resistance. It would typically be combined with traditional encryption and digital signature systems for security, and those systems could use asymmetric keys, so key management would be practical. However, BBC is very recent, and no security analysis of it has ever been published.

This paper is structured as follows. Section 2 describes BBC encoding and decoding. Section 3 then proves its security in the random oracle model. Section 4 then extends that proof to a k-universal hash model. Unlike the random oracle proof, the latter proof applies to systems that can actually be implemented in practice. Section 5 then proves that BBC is not secure for certain large families of hash functions, and gives methods for breaking it in those cases. Finally, section 6 poses several open questions about the security of certainly types of simple hashes.

2. BBC Encoding and Decoding

In jam resistance, the communication channel can be modeled as an OR channel. For example, in a pulse-based system, this means that if Alice wants to send Bob a 1000-bit message during a period of 1 second, which can be divided into one billion time slices, then her communication can be viewed as a vector of a billion bits, most of which are zero, but a few of which are 1 bits. Each 1 denotes a point in time at which she broadcasts a very short, very powerful burst of radio noise that spans a wide range of the spectrum. The jammer James can also broadcast a vector of a billion bits, only a few of which are set to 1. Bob then receives the bitwise OR of the two packets, and must try to decode it.

Such a channel can't be *jam proof*: if James sends a packet of all 1s, then Bob cannot receive any useful information from Alice. However, there is an energy cost to send each pulse. The goal of jam resistance is to force the attacker to use far more energy than the legitimate sender. In this example, if Alice sends 1000 pulses during that second, and if the encoding prevents James from jamming her unless he sends several million pulses, then it is said that the system is successful in resisting jamming. Typical jam-resistant systems in current use have a ratio of a few thousand: James must expend perhaps 4000 times as much energy as Alice (or be closer to Bob by a factor of the square root of 4000) in order to jam the signal.

BBC is the first system to allow jam resistance on an OR channel without a shared secret. Rather than being based on steganography, like current systems, it is based on a new field of coding theory known as *concurrent codes*. This is related to well-studied codes such as *superimposed codes*. A great deal of research has been done related to binary codewords that are combined with a bitwise OR, dating back at least as far as 1956 [4, 5], and being formalized as early as 1964 [6]. If several codewords are combined with a bitwise OR, it is desirable that no other codeword should be contained within that combination. Codes to achieve this in all (or most) cases are variously known as *superimposed codes* [6, 7, 8, 9], *Bloom filters* [10, 11], *strongly selective families* [12], and *cover-free families* (or *r-cover-free families*) [13, 14, 15, 16, 17]. These general terms are all roughly synonymous, though many papers extend and constrain them in various ways, to get categories such as (s,l)-superimposed codes or other generalizations [18, 19, 20, 21, 22, 23, 24, 25, 26].

Typically, these codes assume a fairly small codebook, such as one codeword per user, so it's more likely to be on the order of 1000 codewords rather than 2^{1000} [27, 28, 29, 30, 31]. *concurrent codes* are defined to be those superimposed codes that can be decoded efficiently. At this time, the BBC algorithm is the only known algorithm for doing so.

The BBC algorithm requires the choice of a hash function $H : \{0,1\}^{0\ldots m} \rightarrow \{0,1\}^{lg(n)}$, meaning that the input to the hash function is a string of any length from 0 to the length of the padded message m, but does not need to be defined for strings longer than m. The output of the hash function is $lg(n)$ bits, which are sufficient to choose one location out of the n bits in the packet. It is assumed that the hash function is chosen by Alice or someone trustworthy, and that the hash function is publically known to Alice and Bob, as well as any attackers.

The general BBC algorithm is given in Algorithm 1 (for sending) and 2 (for receiving). The broadcast algorithm tells how to encode and send a single message. The decoding algorithm assumes that multiple, encoded messages were sent simultaneously, combining their binary strings with a bitwise OR, to form a combined string known as a *packet*. In addition, random noise (or the results of an active attack) will flip some of the packet's bits from 0 to 1 (but never from 1 to 0). Under these assumptions, the packet will have the same number of bits as a codeword, but typically many more of the bits will be 1 in the packet than in a single codeword.

The constant k controls the number of checksum bits to use (the bits themselves are simple zeros, but they force the hash function to create a strong checksum). The hash function H maps an arbitrary-length bit string to a location. The general BBC algorithm requires the making of *indelible marks* at *locations* chosen by hashes of all prefixes of the message to be sent. The exact definition of *indelible mark* and *location* depends on the system being used, but the requirement is that both the sender and attacker can make such marks, but neither can erase them. So it is equivalent to a long string of bits that are initially all zero, and the sender and attacker can each change any 0 to 1, but cannot change a 1 to 0. The attacker could jam the communication by setting all the bits to 1, but if there is an energy cost associated with each 1 bit, and if the legitimate sender is setting only a thousandth or a millionth of the bits to 1, then the attacker will be forced to expend a thousand or million times more energy than the sender.

Algorithm 1 BBCbroadcast(M)

This function broadcasts an m-bit message $M[1\ldots m]$ adding k checksum bits to the end of the message. H is a hash function. The definition of H and the values of m and k are public (not secret). The definition of "indelible mark" and "location" are specific to the physical instantiation of BBC used.

Append k zero bits to the end of M
for $i \leftarrow 1\ldots m+k$ **do**
  Make an *indelible mark* at the *location* given by $H(M[1\ldots i])$
end for

Perhaps the simplest instantiation of BBC is when pulses are used, as in the most recent Ultra Wide Band (UWB) systems. The sender broadcasts very short, high-power bursts of radio frequency noise at certain times. The message is encoded in the timing. An attacker can also broadcast such pulses, but cannot erase any existing pulses. Such pulses are difficult to erase because they are very short, very high power, and consist of unpredictable random noise with energy spread over the entire spectrum. No known system exists that can detect and analyze such pulses, and then send out an inverse waveform to cancel them. There simply isn't time during the brief period it takes for the pulse to pass the attacker.

Figure 1 shows how to broadcast using *BBC-Pulse*, an implementation of BBC for pulse-based broadcast. In this example, the message $\mathbf{M = 1011}$ is padded with $k = 2$ zero bits to get 101100. All prefixes of this are hashed with the hash function defined by the table on the right. A pulse is sent at the time defined by the hash of each prefix of 101100. For example, $H(1) = 21$, so a pulse is sent at time 21, and $H(10) = 9$, so another pulse is sent at time 9.

Algorithm 2 BBCdecode(n)

This recursive function can be used to decode all the messages found in a given packet by calling BBCdecode(1). There must be a global $M[1 \ldots m+k]$ which is a string of $m+k$ bits. The number of bits in a message is m, and the number of checksum zeros appended to the message is k. H is a hash function. The definition of H and values of m and k are public (not secret). The definition of "indelible mark" and "location" are specific to the physical instantiation of BBC used.

if $n = m + k + 1$ **then**
  output "One of the messages is:" $M[1 \ldots m]$
else
  if $n > m$ **then**
    $limit \leftarrow 0$
  else
    $limit \leftarrow 1$
  end if
  for $i \leftarrow 0 \ldots limit$ **do**
    $M[n] \leftarrow i$
    if there is an *indelible mark* at *location* $H(M[1 \ldots n])$ **then**
      BBCdecode(M,$n+1$)
    end if
  end for
end if

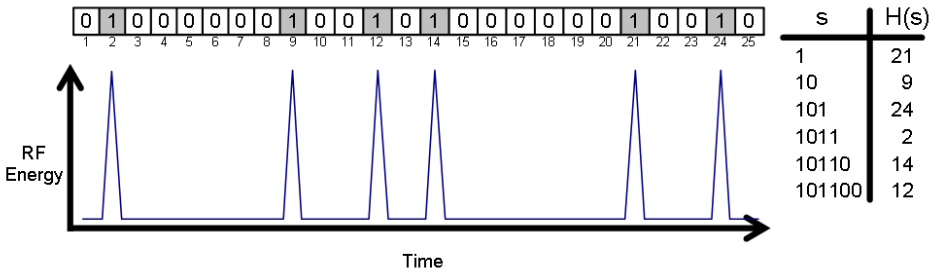

FIGURE 1. BBC broadcast using UWB pulses. $M = 1011$, $k = 2$, an *indelible mark* is a radio pulse, and its *location* is the time when the pulse occurs relative to the start of the message. The table on the right shows part of the definition of the hash function $H(x)$.

Figure 2 shows the decoding of that same message for BBC-pulse. The receiver starts at the root of the tree (on the far left), which is an empty string. The receiver then repeatedly tries appending both a 0 and 1 bit to each string being considered, and calculates the hash of each new string generated. So on the first round, the receiver calculates H(0)=4 and H(1)=21. Since a pulse was not received at time 4, the receiver knows that no messages starting with 0 could have been sent. Since a pulse was received at time 1, the receiver knows that at least one message was sent that started with 1. These results are shown on the tree by coloring the 0 box white and the 1 box gray.

This process then continues. The string 0 is not expanded because no pulse was detected at time H(0). The string 1 is expanded both ways to get 10 and 11, and the receiver calculates H(10) and H(11). Both of these hashes yield times at which pulses were received, so both boxes are colored gray, and both will be expanded on the next round. This process continues until the entire 4-bit message has been decoded (at the dotted line). Note that at this point, both of the legitimate messages have been found (1011 and 1000), but an additional "message" has also been found: 1110. This third message is a *hallucination*.

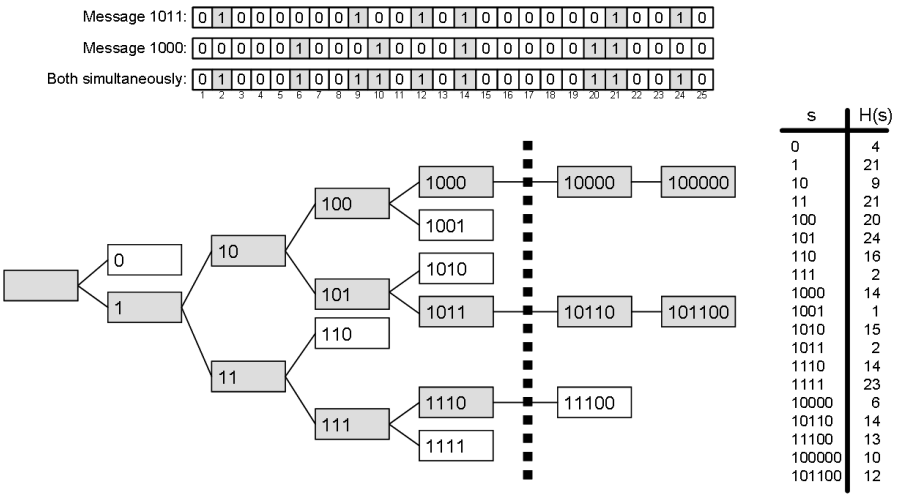

FIGURE 2. Decoding tree for *BBC-Pulse* broadcasts. In this example, both of the messages 1011 and 1000 were broadcast simultaneously, and $k = 2$. The table on the right shows part of the definition of the hash function $H(x)$. The 3rd row at the top shows at which times a pulse was sent (gray boxes) or not sent (white). This is essentially a bitwise OR of the pulse pattern for each message. The binary tree shows those prefixes that are considered during decoding. A prefix is gray if its hash gives a time at which a pulse was detected. The dotted line shows the transition from decoding the body of the message to checking the appended checksum zeros.

This hallucination occurred because all three of the strings 11, 111, and 1110 hashed to locations that just happened to contain a pulse for other reasons. This is why the k checksum bits are important. To the left of the dotted line, every box is expanded by appending both 0 and 1. To the right of the line, only 0 bits are appended. This is because every legitimate message is known to have k zero bits at the end. Both of the legitimate messages survive through all k rounds of checking. The hallucination, however, is eliminated because the string 11100 does not hash to a location that happens to contain a pulse.

This system is intended to be used in situations where a very small fraction of the time positions contain pulses. But consider a worst case, where a full third of all positions contain pulses. If a good cryptographic hash function is used, then these pulses will be scattered pseudorandomly. Then during the checksum process, at each step a given hallucination will have a probability of only 1/3 of surviving, because its hash will choose a pseudorandom location that has a probability of only 1/3 of containing a pulse. If there are k checksum bits, then the probability of a hallucination surviving until the end is only $(1/3)^k$, which quickly becomes vanishingly small for even moderate values of k. The number of potential hallucinations will be small (as is proved in a later section), therefore, the receiver is very unlikely to end up with any hallucinations at all. This, of course, assumes only random noise and accidental interference between legitimate messages. The analysis of an active attack is more complex, and is given in the next section.

3. Security in the Random Oracle Model

If a jammer James wants to stop Alice's communication, he can do so by broadcasting an *attack packet* consisting of all 1 bits. The packet will be bitwise ORed with Alice's packet, preventing Bob from receiving any information whatsoever. However, that requires far more energy than Alice originally used to send the message. If Alice only set 1000 bits to 1 out of a billion bits, then James is forced to expend a million times more energy than Alice used. In traditional jam resistance, a ratio of even a few thousand is considered successfully "jam resistant".

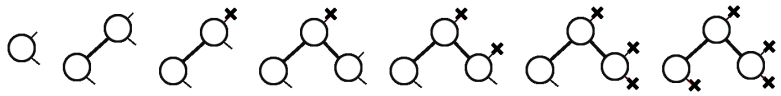

FIGURE 3. The stages of an example BBC decoding that yields a final tree of 3 nodes. Note that the tree ends up with T nodes if there are $T-1$ hashes that map to locations with marks, and $T+1$ that map to locations without marks, and that the Tth Catalan number gives the number of possible trees of size T.

Alternatively, James could send an attack packet consisting of several messages superimposed. If his messages plus Alice's fill up a fraction d of the n bits in the packet with 1s, then Bob will be forced to calculate about dn hash pairs during decoding. Thus, an energy cost of about dn for James yields a computational cost of about dn for Bob. Nothing could be more jam resistant than this, since obviously Bob must decode every validly-formatted message that is sent.

But can James do better? Is there some way that James can set only dn of his bits to 1, yet force Bob to calculate kdn hash pairs for some large k? In other words, can James cleverly design his attack packet so that he gets an advantage of k in computation per energy? Note that the k in this analysis is a different variable than the k used for padding bits in the definition of the algorithm.

Clearly, if James can achieve a k of millions or billions, then he can overwhelm simple processing equipment on Bob's end with a fairly small energy cost to himself. In that case, we would say James succeeded, and the system is not jam resistant. On the other hand, if James can only achieve a k of 2, then he isn't gaining much of an advantage over a simple brute force attack, and the system can be considered jam resistant.

We will now analyze the case where Alice chooses the hash function, James knows what function she chose, and he has unlimited computational power to craft an attack packet that is optimal for attacking that hash function. If Alice chooses the hash function at random (i.e. a random oracle), what is the probability that it will contain a flaw that allows James to achieve a high k value? The following theorem will show that for reasonable parameters, it is *extremely* unlikely that James can even achieve a k of 1.5, even if he has infinite computational power.

Theorem 3.1. *If hash function $H : \{0,1\}^{0\ldots m} \rightarrow \{0,1\}^{lg(n)}$ is chosen randomly and uniformly from the space of all possible hash functions of that size, then the probability that BBC decoding a packet of n bits with a fraction d of them set to 1 (with at least 19^2 bits set to 1) will require more than kdn hash pairs to be calculated is bounded above by:*

$$\left(\frac{1}{1-d}\left(\frac{1}{d}\left(4d\right)^k\right)^d\right)^n \tag{3.1}$$

Proof. We are interested in bounding the probability that a randomly-chosen hash function has *at least one* successful attack packet, where a *successful* attack packet is one that achieves an advantage of k or more. But first, we will analyze a simpler question: what is a bound on the probability that a *randomly-chosen* packet of density d will happen to be a successful attack packet for a randomly-chosen hash function.

Figure 3 shows an example of a decoding process that yields a tree of 3 nodes. Note that it shows the first and third hashes mapped to location in the packet with a 1 bit (thus generating a child node), and that the other 4 hashes mapped to locations with 0 bits (thus generating "X" marks in this diagram, and not generating child nodes). In general, the only way to get a tree with T nodes is for $T-1$ of the hashes to hit a 1 bit, and $T+1$ of the hashes to hit 0 bits. If the hash function is a random oracle, then each time a string is hashed, it will randomly choose a location in the packet to look. Note that BBC decoding of a packet never hashes the same string twice, so all hashes will be independent.

So if the packet has a fraction d of its bits set to 1, then the probability of a particular T-node tree being generated is simply $d^{T-1}(1-d)^{T+1}$. The number of possible trees containing exactly T nodes is the Tth Catalan number, which is $\binom{2T}{T}/(T+1)$. So the total probability of the decode tree containing exactly T leaves is:

$$Prob(nodes = T) = \frac{d^{T-1}(1-d)^{T+1}}{T+1}\binom{2T}{T} \tag{3.2}$$

The probability that the tree has *at most* k nodes will be the sum of this expression as T goes from 1 to k. The probability that the tree has *more than* k nodes will be 1 minus that sum:

$$\begin{aligned} & Prob(nodes > k) && (3.3)\\ & = Prob(\text{this is a successful attack packet}) && (3.4)\\ & = 1 - \sum_{T=1}^{k} \frac{d^{T-1}(1-d)^{T+1}}{T+1}\binom{2T}{T} && (3.5)\end{aligned}$$

This gave the probability that a particular, *randomly-chosen* attack packet was able to break a particular randomly-chosen hash function. But we are actually interested in the probability that *any* attack packet at all exists which breaks a particular randomly-chosen hash function. This latter probability appears difficult to calculate. But there is a simple upper bound on it which can be easily calculated.

We would like to count the number of *bad* hash functions, which are hash functions for which at least one attack packet exists. That is difficult. But it is easy to count the number of (hash function, packet) pairs that are bad. Just multiply the probability of a randomly-chosen pair being bad, by the number of pairs in existence:

$$\begin{aligned} & (\text{number of bad pairs}) && (3.6)\\ & =(\text{number of } n\text{-bit packets of density } d) && (3.7)\\ & \quad \cdot (\text{number of hash functions}) && (3.8)\\ & \quad \cdot Prob(\text{a random pair is bad}) && (3.9)\end{aligned}$$

Clearly, some hash functions will have many bad pairs associated with them. For example, the constant-zero function, which hashes every string to the all-zeros vector, will have an enormous number of attack packets that break it. Any packet which has a 1 in the first position will break it. But we could get an upper bound on the number of bad hash functions by assuming that every bad pair is associated with a different hash function. That, of course, will grossly overestimate the number of bad hash functions. The single example of the constant-zero hash function generates exponentially-many bad pairs, which then inflate our count of bad hash functions by an exponential amount. And it is easy to generate exponentially-many other hash functions that are equally bad, and will also inflate our count by exponential amounts.

However, it turns out that even this hugely-overestimated upper bound will be tight enough to prove the result of this theorem. So this gives:

$$\begin{aligned} & Prob(\text{a randomly-chosen hash function is bad})\\ <& \frac{(\text{number of bad pairs})}{(\text{number of hash functions})}\\ =& \frac{(\text{number of packets})(\text{number of hash functions})Prob(\text{a random pair is bad})}{(\text{number of hash functions})}\\ =& (\text{number of packets})Prob(\text{a random pair is bad})\\ =& \binom{n}{dn}\left(1 - \sum_{T=1}^{k} \frac{d^{T-1}(1-d)^{T+1}}{T+1}\binom{2T}{T}\right)\\ =& \binom{n}{dn}\left(\frac{2^{2k+2}(d-1)^2\Gamma\left(k+\frac{3}{2}\right)\ {}_2F_1\left(1,k+\frac{3}{2};k+3;-4(d-1)d\right)\left((1-d)d\right)^k}{\sqrt{\pi}\Gamma(k+3)} + \right.\\ & \qquad \left. \frac{2d+\sqrt{(2d-1)^2}-1}{2d^2}\right)\end{aligned} \tag{3.10}$$

This expression can be further simplified to a looser upper bound by using known bounds on the binomial function, Γ function, and hypergeometric function ${}_2F_1$ [32]. Useful bounds are:

$$\begin{aligned}&\sqrt{2\pi}\lfloor x-1\rfloor^{\lfloor x-1\rfloor+1/2}e^{-\lfloor x-1\rfloor+\frac{1}{12\lfloor x-1\rfloor+1}}\\ \leq&\Gamma(x)\\ \leq&\sqrt{2\pi}\lceil x-1\rceil^{\lceil x-1\rceil+1/2}e^{-\lceil x-1\rceil+\frac{1}{12\lceil x-1\rceil}}\end{aligned} \tag{3.11}$$

$$\binom{n}{dn}=\frac{\Gamma(n+1)}{\Gamma(dn+1)\Gamma(n-dn+1)} \tag{3.12}$$

$$\frac{1}{1-\frac{xa}{b}}\leq{}_2F_1(1,a;b;x)\leq\frac{1}{1-\frac{xa}{b-1}} \tag{3.13}$$

The remainder of the proof consists of substituting bounds into Equation 3.10, assuming $n>1$, $nd>19^2$, and $d<1/2$, simplifying, making the bounds even looser, noting that the first exponential is less than 1, and the second is less than e^2 (for $n>1$ and $nd>1$).

$$\begin{aligned}&\binom{n}{dn}\left(\frac{2^{2k+2}(d-1)^2\Gamma\left(k+\frac{3}{2}\right)\,{}_2F_1\left(1,k+\frac{3}{2};k+3;-4(d-1)d\right)((1-d)d)^k}{\sqrt{\pi}\Gamma(k+3)}+\right.\\ &\left.\frac{2d+\sqrt{(2d-1)^2}-1}{2d^2}\right)\end{aligned} \tag{3.14}$$

$$\begin{aligned}<&\frac{1}{\sqrt{2\pi}}e^{-\frac{1}{12dn+1}-\frac{1}{12(n-dn)+1}+\frac{1}{12n}}n^{n+\frac{1}{2}}(dn)^{-dn-\frac{1}{2}}(n-dn)^{dn-n-\frac{1}{2}}\\ &\cdot\left[\frac{4^{dkn+1}(d-1)^2e^{-\frac{1}{12(dkn+2)+1}+1+\frac{1}{12(dkn+1)}}(dkn+1)^{dkn+\frac{3}{2}}(dkn+2)^{-dkn-\frac{5}{2}}((1-d)d)^{dkn}}{\left(\frac{4(d-1)d\left(dkn+\frac{3}{2}\right)}{dkn+2}+1\right)\sqrt{\pi}}\right.\\ &\left.-\frac{2d^2-2d+1}{2d^2}+\frac{\sqrt{(2d-1)^2}}{2d^2}+1\right]\end{aligned} \tag{3.15}$$

$$\begin{aligned}<&\frac{1}{\sqrt{2\pi}}n^{n+\frac{1}{2}}(dn)^{-dn-\frac{1}{2}}(n-dn)^{dn-n-\frac{1}{2}}\\ &\cdot\left[\frac{4^{dkn+1}(d-1)^2e^2(dkn+1)^{dkn+\frac{3}{2}}(dkn+2)^{-dkn-\frac{5}{2}}((1-d)d)^{dkn}}{\left(\frac{4(d-1)d\left(dkn+\frac{3}{2}\right)}{dkn+2}+1\right)\sqrt{\pi}}\right.\\ &\left.\frac{-1+2d+\sqrt{(-1+2d)^2}}{2d^2}\right]\end{aligned} \tag{3.16}$$

$$\begin{aligned}<&\frac{e^2}{\sqrt{2\pi}}n^{n+\frac{1}{2}}(dn)^{-dn-\frac{1}{2}}(n-dn)^{dn-n-\frac{1}{2}}\\ &\cdot\frac{4^{dkn+1}(d-1)^2(dkn+1)^{dkn+\frac{3}{2}}(dkn+2)^{-dkn-\frac{5}{2}}((1-d)d)^{dkn}}{\left(\frac{4(d-1)d\left(dkn+\frac{3}{2}\right)}{dkn+2}+1\right)}\end{aligned} \tag{3.17}$$

$$\begin{aligned}=&\frac{e^2}{\sqrt{2\pi}}n^{n+\frac{1}{2}}(dn)^{-dn-\frac{1}{2}}(n-dn)^{dn-n-\frac{1}{2}}4^{dkn+1}(d-1)^2(dkn+1)^{dkn+\frac{3}{2}}\\ &\cdot(dkn+2)^{-dkn-\frac{5}{2}}((1-d)d)^{dkn}\frac{2+dkn}{2+dkn+4(-1+d)d\left(\frac{3}{2}+dkn\right)}\end{aligned} \tag{3.18}$$

$$= \frac{4e^2}{\sqrt{2}\pi} \frac{1}{2 + dkn - 4(1-d)d\left(\frac{3}{2} + dkn\right)} (dn)^{-\frac{1}{2}} \left(\frac{1}{1-d}\frac{1+dkn}{2+dkn}\right)^{\frac{3}{2}} \cdot \left(\frac{1}{1-d}\left(\frac{1-d}{d}\left(4d(1-d)\left(\frac{1+dkn}{2+dkn}\right)\right)^k\right)^d\right)^n \tag{3.19}$$

$$< \frac{19}{\sqrt{dn}} \left(\frac{1}{1-d}\left(\frac{1}{d}(4d)^k\right)^d\right)^n \tag{3.20}$$

$$< \left(\frac{1}{1-d}\left(\frac{1}{d}(4d)^k\right)^d\right)^n \tag{3.21}$$

Thus, the probability is bounded above by the desired expression, so the theorem is proved. □

Note that this theorem implies that in a typical case such as $n = 10^9$ and $d = 10^{-3}$, then the probability of a randomly-chosen hash function allowing even a single attack packet with an advantage for the attacker of $k = 1.5$ is guaranteed to be less than $10^{-162339}$, which is an exceedingly small probability. But for $k = 1.4$, the theorem gives an upper bound on the probability that is greater than 1.0, and so tells us nothing. This is typical behavior for the bounding function. The expression inside the outer perentheses is raised to the power of n, which is typically a very large number. So if that expression is less than 1.0, then the overall bound will typically be very close to zero. If the expression is 1.0 or more, then the bound tells us nothing. It is therefore useful to ignore the n, and simply solve for the value of k that makes that expression exactly equal to 1.0. Solving for k gives:

$$k = log_{4d}(d(1-d)^{1/d}) \tag{3.22}$$

In other words, given the density d that the attacker can afford to create in the attack packet, as long as the hash function was chosen at random, it is almost certain that the attacker will be unable to force Bob to do more than a factor of k additional computation beyond the brute force attack. And this is true even if the attacker has infinite computational power to look for the optimal attack packet for that particular hash function. Better attack packets simply won't exist, with high probability. For typical numbers, this k is less than 1.5, so Bob will have to do less than 50% additional compution. Therefore, in this example, if Bob is able to perform 50% more computation, in addition to computation proportional to the energy limitations of the attacker, then the BBC algorithm is shown to be unconditionally secure with high probability in the random oracle model.

4. Security in the k-Universal Hash Model

The theorem in the previous section shows that BBC is secure in the random oracle model. Unfortunately, random oracles cannot be implemented in the physical world, so that does not prove the security of BBC for any particular choice of hash function. Although it might increase our confidence in BBC implemented using, say, SHA-1, it is still possible that it might fail.

The following theorem solves this problem by proving the same bounds under weaker assumptions on the hash function. In fact, it allows families of hash functions that can choose a function at random using only polynomial time and space, and can also evaluate that function in polynomial time and space.

Theorem 4.1. *The bounds of Theorem 3.1 also hold if the hash function is chosen uniformly from a family of $2kdn$-universal hash functions, rather than from the space of all possible hash functions of a given size.*

Proof. Note that the proof of Theorem 3.1 involved first generating random (hash function, packet) pairs, and finding the probability that such a pair will require more than kdn hash pairs to be calculated. The analysis of that step assumed that when the hash function is evaluated $2kdn$ times, it acts like a random hash function. That is, if the function is chosen randomly and is unknown, then knowledge of its behavior on the first $i-1$ evaluations should give no information about its behavior on the ith evaluation, for all $i \leq 2kdn$. This assumption is valid in the random oracle model, where the hash function is chosen randomly

from the space of all hash functions of a given size. The assumption is also valid for a $2kdn$-universal hash, by definition.

A k-universal hash is a family of hash functions such that, if one is chosen at random, it will behave like a random oracle the first k times it is evaluated, regardless of what inputs it is evaluated on. If it is evaluated more than k times, then there can be correlations or dependencies among its results that can distinguish it from a true random oracle. But that will not be possible for the first k evaluations.

There are many k-univeral hashes known that handle inputs up to m bits, and which use time and space polynomial in k and m. One simple example is a kth-order polynomial modulo an m-bit prime, where the $k+1$ coefficients are chosen randomly and uniformly from the space of all nonnegative integers less than the modulus. It takes only polynomial time and space to generate and store such a polynomial, and only polynomial time to evaluate it on a particular m-bit string.

Therefore, if such a $2kdn$-universal hash is used in BBC to map strings of up to m bits to outputs of $lg(n)$ bits, the proof in Theorem 3.1 goes through without change, and the probability bounds proved there are still correct. □

5. Insecurity of Small Internal State

The hash function in BBC maps a string of at most m bits to an output of only $lg(n)$ bits. For example, it might map from at most 1000 input bits to 30 output bits. In that case, one might be tempted to use a simple hash function that maintains only 30 bits of internal state. However, that is provably insecure, regardless of how the hash is constructed.

Theorem 5.1. *If a hash function $H : \{0,1\}^{0...m} \rightarrow \{0,1\}^{lg(n)}$ has only S bits of internal state (i.e. it consumes the input string bit by bit and updates the S bits of internal state after each bit), then it is possible to construct an attack packet that will cause workload k for the receiver that is exponential in m/S.*

Proof. This proof is by construction. An attack packet will be created that has at most $2m$ one bits set, yet it will cause the receiver to perform at least $2^{m/S}$ computations.

First, find two different strings, A_1 and B_1, each of length m/S, such that the internal state of the hash function is S_1 after reading in either string. If the two messages are the same length as the internal state of the hash, then such a collision must exist.

Next, find two different strings A_2 and B_2, such that the internal state after reading A_1 followed by A_2 is S_2, as is the state after reading B_1 followed by B_2. In other words, the string found by concatenating A_1 with A_2, and the string found by concatenating B_1 and B_2 will each cause the state to reach S_1 followed a short time later by S_2.

Continue this process, finding strings A_i and B_i for all i until two long strings of length m have been found. BBC encode those two strings, and bitwise OR the two packets together to form the attack packet.

Note how much computation the receiver must perform to decode this attack packet. The packet will obviously contain the two messages that were sent:

$$(A_1, A_2, A_3, A_4, ..., A_{m/S}) \tag{5.1}$$

$$(B_1, B_2, B_3, B_4, ..., B_{m/S}) \tag{5.2}$$

But it will also contain any message found by swapping segements from the two messages. For example, it will contain

$$(A_1, B_2, A_3, B_4, ..., A_{m/S}) \tag{5.3}$$

That is because the internal state of the hash collides m/S times in the course of these strings, and so the message formed by swapping will have all of its hashes map to the same locations as the original two messages. Therefore, there are $2^{m/S}$ different messages contained in the packet, all of which must be decoded. For example, if the 1000-bit message is encoded using a hash with a 30-bit internal state, then there will be 2^{33} messages to decode, requiring $(1000)(2^{33})$ hash pairs to be calculated. If this is too much for the receiver, then the attacker has jammed the receiver with very little energy cost, merely twice that of sending a legitimate message.

Note, too, that if $S = 30$, there are only a billion possible internal states for the hash function, and each collision can be found in 2^{15} steps on average with a birthday attack. Thus, this attack is extremely efficient, even for large packets of a billion bits. □

6. Open Questions: Linear Hashes

We complete our analysis of the security of BBC by posing two open questions. These deal with the security of linear hash functions.

There are two simple ways to define a linear hash function. One is to define the hash of a b-bit string x to be the $lg(n)$-bit string y defined as:

$$y = xA^{b,lg(n)} mod 2 \tag{6.1}$$

where x and y are treated as row vectors of bits, $A^{b,lg(n)}$ is a b by $lg(n)$ matrix of bits that were generated randomly and uniformly, and all the calcualtions in the matrix multiplication are done modulo 2. In this model, a different A matrix must be generated independently for each string length from 0 up to m.

A second way to define a hash function is with a linear feedback shift register with m bits of internal state. The register is initialized to a fill of all 1s. The string is then read in by cycling the register one time for each bit in the string. After each shift, the corresponding bit of the string is XORed into the state of the register at one or more locations. After the string has been read in, the $lg(n)$-bit hash is generated by a linear function of the state of the register.

This gives two different ways of defining a linear hash function. The second method is a subset of the first, and can be implemented using less memory. Neither of these hashes are $2kdn$-universal, so the security proofs don't apply. Neither of these hashes have small internal state, so the insecurity proof also doesn't apply. But the linear nature of these hashes should make them easy to analyze. So is BBC secure using either or both of these methods? We leave this as an open question for further research.

7. Conclusions

We have proved the unconditional security of BBC against an adversary with unlimited computation, but limited energy (i.e. a limit on how many zeros can be set to ones during the attack). In both the random oracle model and the k-universal hash model, we have shown that a randomly-chosen hash function will be secure with extremely high probability. We have also proved that BBC with a hash function with limited internal state is insecure, and have given an algorithm for breaking it. Given these results, it appears that BBC is a useful primitive for protocols that require jam-resistant broadcast.

Several interesting open questions remain. We know that almost all modular polynomials are secure, because they form a k-universal hash. But we don't know whether any particular polynomial is secure. Is there a way to find one? Is multiplication by a random matrix (with a different matrix for each possible string length) a secure hash function for BBC? Is a random linear feedback shift register secure? These appear to be fruitful areas for further research.

References

[1] http://www.globalgadgetuk.com/Personal.htm.

[2] L. Baird, B. Bahn, and M. Collins. Jam-resistant communication without shared secrets through the use of concurrent codes. Technical Report USAFA-TR-2007-01, U.S. Air Force Academy, February 2007.

[3] L. Baird, B. Bahn, M. Collins, M. Carlisle, and S. Butler. Keyless jam resistance. In *Proceedings of the 8th Annual IEEE SMC Information Assurance Workshop (IAW)*, June 2007.

[4] M. Taube. Superimposed coding for data storage. Technical Report Rept. No. 15.

[5] F. Rogers. [a review of] studies in coordinate indexing [by mortimer taube]. *Bull Med Libr Assoc.*, 42(3):380–384, July 1954.

[6] W. Kautz and R. Singleton. Nonrandom binary superimposed codes. *IEEE Transactions on Information Theory*, pages 363–377, 1964.

[7] D. Danev. Some constructions of superimposed codes in euclidean spaces. *Discrete Applied Mathematics*, 128(1):85–101, May 2003.

[8] G.T Ahlstrm and D Danev. A class of superimposed codes for cdma over fiber optic channels. Technical Report LiTH-ISY-R-2543.

[9] Peter-Olof Anderson. Superimposed codes and bn-sequences. Technical Report LiTH-ISY-I-1108.
[10] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, July 1970.
[11] Haoyu Song, Sarang Dharmapurikar, Jonathan Turner, and John Lockwood. Fast hash table lookup using extended bloom filter: an aid to network processing. In *SIGCOMM '05: Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 181–192, New York, NY, USA, 2005. ACM Press.
[12] A. Clementi, A. Monti, and R. Silvestri. Distributed broadcast in radio networks of unknown topology. *Theor. Comput. Sci.*, 302(1-3):337–364, 2003.
[13] Z. Furedi Z. Furedi P. Erdos, P. Frankl. Families of finite sets in which in which no set is covered by the union of r others. *Israel Journal of Mathematics*, 51:75–89, 1985.
[14] M. Ruszinko. On the upper bound of the size of the r-cover-free families. In *Proceeding of the 1993 IEEE International Symposium on Information Theory*, page 367, 1993.
[15] Arkadii D'yachkov, Vladimir Lebedev, Paul Vilenkin, and Sergi Yekhanin. Cover-free families and superimposed codes: Constructions, bounds, and applications to cryptography and group testing. In *IEEE International Symposium on Information Theory*, 2001.
[16] D. R. Stinson and R. Wei. Generalized cover-free families. *Discrete Mathematics*, 279:463–477, 2004.
[17] R. Wei. On cover-free families. Technical report, Lakehead University, 2006.
[18] Arkadii D'yachkov, Anthony Macula, David Torney, Pavel Vilenkin, and Sergey Yekhanin. New results in the theory of superimposed codes. In *roceedings of International Conf. on Algebraic and Combinatorial Coding Theory (ACCT)*, pages 126–136, 2000.
[19] Sergy Yekhanin. Sufficient conditions of existence of fix-free codes. *IEEE International Symposium on Information Theory*, June 2001.
[20] A. Dyachkov and V. Rykov. Optimal superimposed codes and designs for renyi's search model. *Journal of Statistical Planning and Inference*, 100:281–302, 2002.
[21] A. de Bonis and U. Vaccaro. Constructions of generalized superimposed codes with applications to group testing and confict resolution in multiple access channels. *Theoretical Computer Science*, 306:223–243, 2003.
[22] A. D'yachkov, P. Vilenkin, and S. Yekhanin. Upper bound on the rate of superimposed (s,l) codes based on engel's inequality. In *Proceedings of the International Conf. on Algebraic and Combinatorial Coding Theory (ACCT)*, pages 95–99, 2002.
[23] S. Yekhanin. Some new constructions of optimal superimposed designs. In *Proceedings of International Conf. on Algebraic and Combinatorial Coding Theory*, pages 232–235, 1998.
[24] A. de Bonis and Ugo Vaccaro. Efficient constructions of generalized superimposed codes with applications to group testing and conflict resolution in multiple access channels. In *ESA*, pages 335–347, 2002.
[25] Thomas Erickson and Laszlo Gyorfi. Superimposed codes in rn. Technical Report LiTH-ISY-I-0818.
[26] Tayuan Huang and Chih wen Weng. A note on decoding of superimposed codes. *Journal of Combinatorial Optimization*, 7:381–384, 2003.
[27] D. Awduche and A. Ganz. Mac protocol for wireless networks in tactical environments. In *IEEE Military Communications Conference, MILCOM-96*, October 1996.
[28] Lars-Inge Alfredsson. A class of superimposed codes for cdma over fiber optic channels. Technical Report LiTH-ISY-I-1045.
[29] Fidel Cacheda and Ángel Viña. Superimposing codes representing hierarchical information in web directories. In *WIDM*, pages 54–60, 2001.
[30] Y. Desmedt. A high availability internetwork capable of accomodating compromised routers. *BT Technology Journal*, 24(4):77–83, July 2006.
[31] Janos Komlos and Albert Greenberg. An asymptotically nonadaptive algorithm for conflict resolution in mutliple-access channels. *IEEE Transactions on Information Theory*, IT-31(2):302–306, March 1985.
[32] D. Karp and S. Sitnik. Inequalities and monotonicity of ratios for generalized hypergeometric function. *J. of Approximation Theory*, Mar:1–14, 2007.