# A New Algorithm for Unkeyed Jam Resistance

Hamid Hanifi
University of Denver
Departement of Computer Science
Denver, Colorado, USA.
hhanifi@du.edu

Leemon Baird
leemon@leemon.com

Ramakrishna Thurimella
University of Denver
Department of Computer Science
Denver, Colorado
ramki@cs.du.edu

## ABSTRACT

An important problem for secure communication is that of achieving jam resistance, without any prior shared secret between the sender and receiver, and without limits on the assumed computational power of the attacker. To date, only one system has been proposed for this, the BBC system, which is based on coding theory using codes derived from arbitrary hash functions. It is unfortunate that only one, narrow solution has been found for this important problem. We now propose a new algorithm for this problem: the HBT algorithm. It is very different from BBC, using codes based on monotone Boolean functions (MBF), rather than hash functions. It is also more general. We show that despite being very different from BBC, the latter can be viewed as a special case of it. In fact, a theorem proves that all such codes are special cases of this new system. We give empirical results suggesting that this new approach is useful, and describe directions for future research.

## Keywords

Jam Resistance, Coding Theory, Wireless Communications, Spread Spectrum

## 1. INTRODUCTION

Communication over noisy channels has been studied extensively. In modern communication, achieving jam resistance is increasingly important, as attackers can easily obtain inexpensive jamming devices on the open market. This is specially critical in military applications as even a brief denial of service attack can severely impact a vital operation. In the past, jam resistance has been achieved through the use of spread spectrum techniques that involve a secret, shared by the sender and receiver, but not by the attacker. This secret might then define a chip sequence (for direct sequence spread spectrum) or a hop sequence (for frequency hopping). Some common examples include[3, 5, 7].



*SIN '15, September 08 - 10, 2015, Sochi, Russian Federation*



ACM ISBN 978-1-4503-3453-2/15/09…$15.00

DOI: http://dx.doi.org/10.1145/2799979.2800008

Concurrent codes were first proposed in [4] (and later in [2]) as a means of achieving jam resistance without a shared secret. Concurrent codes are superimposed codes that can be decoded in polynomial time with high probability. That is, if a *packet* is formed by taking either a single codeword or the bitwise OR of several codewords, and it is then corrupted by an attacker who changes several of the 0 bits to 1, then there is an efficient algorithm that allows the receiver to quickly find all of the codewords contained in that packet. The advantage of this technique is that it does not require a shared secret.

## 2. BACKGROUND

Spread spectrum techniques have been in use since the 1940s to achieve resistance to jamming, natural noise, and eavesdropping. They can be based on direct sequence, frequency hopping or pulse-based approaches. In these systems, it is often possible to achieve *jam resistance*, which means the attacker must spend far more energy than the sender, in order to prevent the receiver from receiving the sent message.

The problem with traditional jam resistance techniques is that they require some shared secret (or *key*) to achieve jam resistance. This is an issue when we try to apply these techniques to large scale applications like GPS, where the intended "users" are everyone on earth, and so include the potential attackers. There is no way to protect GPS signals using these techniques as these signals are meant to be public. Even for private networks, if there are many devices that must communicate in an ad hoc network, and if an attacker might physically capture one of the devices, then using a shared secret is problematic.

We need a system for omnidirectional radio communication that achieves jam resistance without any shared secret, and without assuming any limit on the attacker's computation power. The only assumed limitation will be on the amount of radio frequency power that the attacker can broadcast (since broadcasting infinitely-powerful noise can jam anything, but requires infinite energy). The BBC algorithm [4][2] introduced the first algorithm for keyless jam resistant communication. BBC uses *superimposed codes*, which were introduced in 1964 by Kautz-Singleton [6]. A superimposed code is defined to be a set of codewords such that when a small number of codewords are combined using a bitwise OR, the result (or *packet*) will include no codeword other than those used to form the sum. The codes that were proposed then were extremely inefficient, and impractical to use for sending messages of reasonable length. The BBC

codes are a generalization of this called *concurrent codes*, which are efficient (polynomial expected time and space to encode and decode) and probabilistically correct. The probabilistic correctness means that when several codewords are ORed together, and several of the 0 bits are changed to 1 bits, then as long as the resulting packet does not have too many 1 bits, the result of decoding will, with high probability, contain all the original codewords that were combined, plus only a small number of additional codewords. These extra codewords are called *spurious codewords*, and can then be filtered out in a later step, using all the standard communication techniques, such as checksums, cryptographic hashes, or digital signatures. For example, if a message is received with an incorrect checksum or digital signature, then it can be discarded.

When a receiver receives all messages sent by various senders, the receiver can recognize messages intended for that receiver by looking at the TO address field, can recognize damaged messages by using checksums, and can recognize forged messages by using signatures. If there are several spurious messages, then they are discarded on the bases of TO fields, checksums, or signatures. So spurious messages are not a problem unless the attacker is able to send a large number of them. In a good jam resistance system, that should require a very large amount of broadcast energy.

BBC was the first concurrent code. Our new HBT algorithm is the second concurrent code that can be used to accomplish keyless jam resistant communication. We use monotone Boolean functions (MBF) and more specifically, monotone Boolean circuits (MBC), to encode and decode messages. We will show that this is an alternative to BBC algorithm for jam resistance communication without a shared secret.

A *codeword* is a fixed-length binary string, and a *codebook* is a set of such strings. The HBT algorithm defines a codebook associated with every *monotone Boolean function (MBF). An MBF is any function* $f : \{0,1\}^n \rightarrow \{0,1\}$ *such that:*

$$\forall x, y \in \{0,1\}^n : f(x) \leq f(x \vee y)$$

In other words, an MBF is boolean, because the inputs are bits and the output is a single bit, and it is monotone because changing any single input from 0 to 1 will either leave the output unchanged, or will change it from 0 to 1 (never from 1 to 0). The HBT algorithm requires that the MBF can be evaluated in polynomial time, and defines the *codebook* to be the set of all *minimal true vectors* (MTV) for that MBF. An MTV is a binary string that when used as input to the MBF makes it output 1 (*true*), but where changing any single 1 bit to 0 will cause the output to change to 0 (*false*).

With this definition, any choice of MBF will define a codebook. And it is easy to create new MBFs. For example, any feedforward circuit consisting only of AND and OR gates will constitute an MBF.

However, not all MBFs will form a useful concurrent code. For example, if most codewords have a large number of 1 bits, then ORing together even a small number of them could result in a packet of all 1s, which would then decode as containing every codeword in the entire codebook! So, to be useful, the codewords will need to be *sparse*, or *low density*, containing mostly 0 bits and only a few 1 bits.

On the other hand, if the average codeword density is too low, then there won't be enough codewords to convey very much information. For example, if the packets are $n$ bits long, and each codeword has only a single 1 bit, then there can be at most $n$ codewords, which is far smaller than the $2^{1000}$ codewords that would be needed if we want a single codeword to communicate a 1000-bit message. This same problem would arise if the codeword density were good, but the number of codewords in the codebook was too small. There is no simple way to look at a family of circuits and guess how large the codebook will be, without testing it empirically.

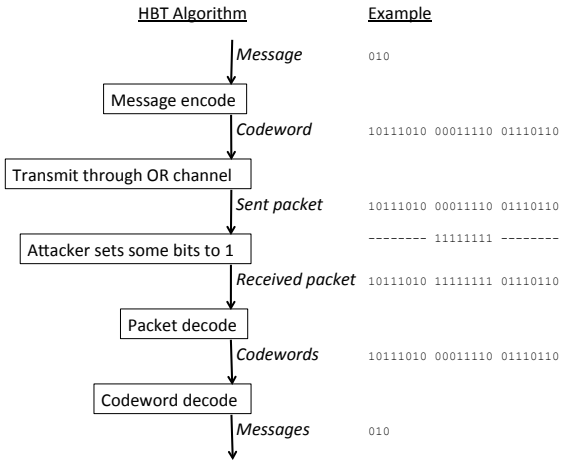


**Figure 1: The HBT algorithm with three sub algorithms (message encode, packet decode, codeword decode)**

Furthermore, the codebook will give excessive spurious messages during decoding if it contains certain patterns and is "non-random" is certain ways. For example, suppose all of the codewords are very similar (separated by low Hamming weight). Then it is likely that the bitwise OR of two codewords will give too many spurious messages, because one codeword can be converted into another by setting only a few bits to 1. It is not at all obvious how "random" the codebook will be for a given MBF or MBC, until it is tested empirically. So we have searched through many types of circuits to find a family of MBCs that implement MBFs which define codebooks that avoid these three problems (too dense or sparse, too many or too few codewords, too "non-random" in the sense of generating too many spurious codewords during decoding). The results in this paper give the first circuit families that were found that give promising results in these areas.

Of course, even if the MBF defines a useful codebook, it may not be obvious how to use that MBF to send and receive messages. That can be done using the HBT algorithm.

# 3. THE HBT ALGORITHM

The HBT algorithm is shown in figure 1. This main algorithm is composed of three sub algorithms that are described later: message encoding (Algorithm 2), packet decoding (Algorithm 1), and codeword decoding (figure 4).

The sender encodes the message to get a codeword, which is sent over an *OR channel*. An OR channel is a communication channel for bits, where usually, a 0 is received if every sender is sending a 0, and a 1 is received otherwise. In such

a channel, it is much easier for an attacker to change a 0 to a 1 than to change a 1 to a zero. An example is given in [1], based on Golay chip sequences. Some cell phone systems currently achieve initial synchronization by broadcasting a pseudorandom chip sequence called a Golay sequence, for which it is particularly easy to build matched filters. This means that a receiver can easily detect that the sequence was sent, and when it was sent, without any initial synchronization. The sequence is spread spectrum, and therefore resistant to noise. It is unmodulated: it contains no information other than the time at which it started. So it can be thought of as transmitting a sequence of all zeros, with a single 1 denoting its start. If two such sequences are sent simultaneously, but with starts slightly offset in time, then the receiver will receive two times, rather than one, and will "see" a sequence of 0 bits with just two 1 bits. By superimposing multiple such sequences, it is possible to send a string of 0 bits with an number of 1 bits. It is very easy for an attacker to change a 0 to a 1: simply broadcast one more copy of the sequence, starting at the appropriate time. But it is very difficult for the attacker to change a 1 to a 0. That would require erasing the fact that a chip sequence was sent at all. Therefore, this constitutes an OR channel.

The bit stream received by the receiver is called a *packet*. It contains the codeword that was sent. If several codewords were sent simultaneously, then the packet would contain the bitwise OR of those codewords. And if an attacker was trying to jam communication, it would have some number of its 0 bits changed to 1. The receiver then finds the complete set of codewords that are contained in that packet, using Algorithm 1. Then, for each codeword, the receiver converts it to a message using the approach shown in figure 4. If multiple messages are sent simultaneously, by a combination of multiple senders and attackers and natural noise, then the receiver finds the desired messages using some combination of address, checksums, and signatures, as described above.

# 4. FINDING A GOOD MBF

It is not obvious how to find a good MBF that will allow long messages to be sent, while generating few spurious messages for the receiver. It needs to have the properties already mentioned: a good density of 1 bits, a good number of codewords, and a sufficiently random distribution of codewords.

We explored a huge number of different families of AND/OR circuits (MBCs) over the course of several years before finding the family shown in this paper. When random circuits are generated from this family, we find that the codewords have good densities, and that we are able to encode messages of reasonable length, and that it gives few enough spurious messages that we are able to quickly check them all for the desired messages.

An example circuit in this family is shown in Figure 2, and the best parameters for various input sizes are shown in Figure 5.

For this family, a circuit of $n$ inputs will have $n$ gates in each layer, with a single gate in the output layer. Gates in one layer are connected to gates in the preceding and following layers. The output gate is an AND gate, and it alternate OR and AND gates in the preceding layers. In a given layer, all gates are the same type, and all have the same number of inputs and the same number of outputs. The behavior is critically dependent on the number of inputs and outputs for each layer, and the best numbers for these circuit sizes are given in Figure 5.

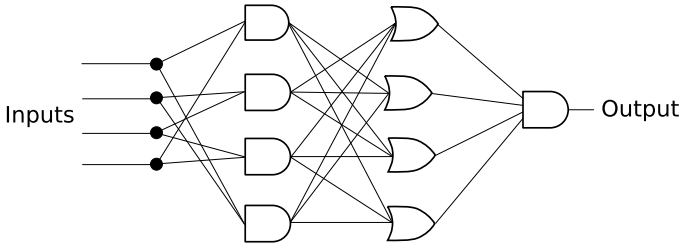


**Figure 2: An MBC with n=4 inputs, m=3 layers**

# 5. CIRCUIT GENERATION

The encoding process includes finding a circuit to create a codebook and then use the codewords from the codebook to create packets by doing bitwise OR on the codewords. Each codeword represents a message. BBC suggests to have packets with no more than 1/3 of the bits set to 1. We try to maintain the same packet density to be able to compare the results with BBC. Also keeping the packet density below 1/3 increases the chance that we would be able to recover the messages at the receiving side despite the jamming attempts by the attackers. As a result, the ideal number of 1s in the codewords should be much less that 1/3 in order to be able to OR the codewords together and create packets with less than 1/3 of bits set to 1.

Defining the circuit includes defining number of rows, layers (columns) and the connection matrix. The connection matrix is populated using the transposing matrix algorithm explained below:

## 5.1 Generating a connection matrix

If two adjacent layers have $n$ gates each, we generate an $n$ by $n$ *connection matrix*, that has $k$ bits set to 1 in each row and each column. This ensures that all gates in the first layer have the same number of outputs, and all gates in the second layer have the same number of inputs. We generate the circuit randomly, subject to this constraint. We found empirically that the behavior is the same every time we generate a new random connection matrix, for any given choice of $n$ and $k$ in all the layers.

To generate an $n$ by $n$ matrix that is all zeros with $k$ 1 elements in each row and column, we first generate each column, as $k$ 1 bits followed by $n - k$ 0 bits. Then randomly permute each column independently, using a standard Durstenfeld shuffle as described by Knuth (i.e., swap each element with itself or an earlier element). Then search from the top row downward, to find the uppermost row with more than $k$ 1 bits and the uppermost row with fewer. Then search from left to right until a column is found where the too-few row has a 0 and the too-many row has a 1. Swap those two elements. Repeat until all rows have the same number of 1 bits, which will then give the desired pattern.

For a given number of circuit inputs $n$, it is necessary to find an appropriate choice of $k$ in between each pair of adjacent layers (where the $k$ for one pair can differ from the $k$ for another pair). This is done by randomly generating circuits with various combinations of choices, and trying the HBT algorithm on random messages to evaluate how well it

works. We found that for these families, the search can use gradient descent, because the error function is smooth.

# 6. FINDING THE CODE BOOK

The HBT packet decoding algorithm (Algorithm 1) is equivalent to the Universal Concurrent Code algorithm introduced in BBC [4] to find codewords in a packet. The algorithm describes the process to find a single codeword in a packet, by visiting the bits in a random order, setting each to 0, and resetting it back to 1 if setting it to 0 makes the output become 0. This probabilistic function returns one codeword from the n-bit packet P. All the codewords can be found by calling it repeatedly until it starts returning only repeats of previously-found codewords. The returned codeword will be an element of the codebook defined by the polynomial-time monotonic Boolean function f. This runs in linear time (plus the time for n calls to evaluate f). Every codeword in the packet has a nonzero probability of being returned, and all other bit vectors have zero probability.

In order to find all the codewords in a codebook, or a random subset of them, we simply decode a packet of all 1 bits, repeatedly. Because the algorithm visits the bits in a random order, it will be the case that each time the all-ones packet is decoded, the result will be a random codeword from the codebook.

**Algorithm 1** HBT Packet Decoding Algorithm

```
Input: Monotone Boolean function f, n-bit packet P
Output: A codeword if the packet contains any.
if f(P) = 0 then
    return The packet contains no codewords
end if
M ← P
L ← the list 1, 2, ..., n in a randomly-permuted order
for i = 1 → n do
    M[L[i]] ← 0
    if f(m) = 0 then
        M[L[i]] ← 1
    end if
end for
return M
```

The total number of codewords in a small codebook can be counted by generating random codewords from it repeatedly, until most of the generated codewords are ones that have already been seen. But this is impractically slow for large codebooks. So we need a way to estimate the codebook size for a given MBF. One way is to estimate it from the *critical density*. This is defined to be a real number $d$ in the range $(0, 1)$, such that when each input bit to the MBF is set to 1 with probability $d$, then the output will be 1 with probability 1/2. Given this definition, we can estimate the size of a codebook by measuring $d$ for a circuit, and the estimate will be good if the implied codebook looks sufficiently random. This result is given in Theorem 1.

**Theorem 1.** *If a random codebook has $N$ codewords of $n$ bits each, with each having $a$ bits set to 1 and the rest set to 0, and a critical density of $d$, then the following equation will hold:*

$$1 - (1 - \frac{N}{C(n,a)})^{C(dn,a)} = \frac{1}{2}$$

*Proof.* There are $C(n, a)$ possible $n$-bit strings with $a$ of the bits set to 1, where $C$ is the binomial function "n choose a". Only a fraction $N/C(n, a)$ of such strings are actual codewords, so a fraction $1-N/C(n, a)$ of them are not codewords. If an input packet is created with the critical density, then there will be $dn$ bits in it set to 1, and so $C(dn, a)$ potential codewords (strings with $a$ bits set to 1) contained within that packet. The probability that all of them are outside the codebook is $(1 - N/C(n, a))$ to the power of $C(dn, a)$. And so the probability of at least one of them being in the codebook is 1 minus that. That final probability must be 1/2, by the definition of "critical density". Therefore the equation shown is correct. □

# 7. MESSAGE ENCODING AND CODEWORD DECODING

Once all the codewords in a received packet have been found, it is necessary to convert each one to a message. This is done using the HBT Codeword Decoding algorithm (figure 4). The codeword is broken into equal-sized *sections*, each of which converts to one bit of the message. Within a section, the Hamming weight (number of 1 bits) is found for each half of the section, and the section is consider to represent a 0 if the first half has greater weight, or a 1 if the second half has greater weight (or the weights are equal). Figure 4 shows an example of the decoding algorithm in action. The table on the top show association of messages to codewords. The left section of the table lists 3-bit messages ready to be encoded into codewords. The right section of the table shows the codewords resulting from algorithm.

Given this method for converting a codeword to a message, how can we go the opposite direction, and convert a message to a codeword? That is shown in Algorithm 2. It works by starting with the all-ones packet, and decoding it to find a single codeword. However, the bits are not visited in a completely random order. Instead, it first visits the half of each section that is meant to be lighter (i.e., fewer 1 bits). Then it visits all the light halves again, and repeats this until it has visited every bit in the light halves. Then it visits the heavy half bits in a random order. Finally, check the resulting codeword to see if it successfully encodes the desired message. If it does not, try again.

This actually works correctly and quickly, if the codebook has about $2^k$ sparse codewords, and the message has at most $k$ bits. That is because we can usually choose about $k$ of the bits to be zero, before we reach the point that there is exactly one codeword in the codebook that matches those constraints. So the first $k$ bits we visit will be set to zero, and the rest will seem to be randomly generated, giving the average density. If the light halves are chosen first, then there will be guaranteed zeros in the light halves, and so with high probability all the message bits will be correct.

This behavior is shown in figure 3. We define the density curve to be the expected number of 1 bits generated in the first t steps, versus t ( $t \leq n$), which is the left graph. We can also numerically take the slope of this curve, which is the probability that a visited bit will be 1 vs. the order in which that bit was visited (right graph). So these two graphs give insight into why the algorithm works.

We define the density curve to be the expected number of 1 bits generated in the first t steps, versus t ( $t \leq n$). For a perfect circuit density curve looks like figure 3.

**Algorithm 2** HBT Message Encoding Algorithm

Algorithm

Input: Message M[k], a MBF
Output: A codeword that encodes message M
$L[k] \leftarrow$ length of each of k group
**for** $i = 1 \rightarrow k$ **do**
  //try to set the desired-0 bits to 0
  **for** $1 \rightarrow L[i]$ **do**
    **if** M[i] =1 **then**
      visit the least significant half of bits in group i, try to set the bits to 0
    **else**M[i]=0
      visit the most significant half of bits in group i, try to set the bits to 0
    **end if**
  **end for**
  //Now set the desired-1 bits to 0
  **for** $1 \rightarrow L[i]$ **do**
    **if** M[i] =0 **then**
      visit the least significant half of bits in group i, try to set the bits to 0
    **else** M[i]=1
      visit the most significant half of bits in group i, try to set the bits to 0
    **end if**
  **end for**
**end for**
**if** (number of 1s in all desired-1 half-sections > number of 1s in desired-0 half-sections) **then**
  **return** codeword
**end if**

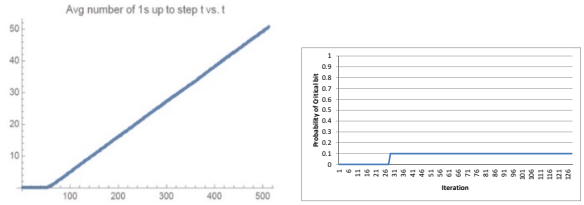


**Figure 3: Density Curve (left) and bit probability (right)**

As we discussed before environmental noise or attackers can add unwanted 1 bits to the packet, but changing 1 bits to 0 bits is not possible as it is equivalent of removing energy from the medium. An attacker needs to consume a considerable amount of energy in order to introduce enough number of unwanted 1 bits in the packet to interfere with the decoding process. This asynchrony is what makes us capable of making our communication jam resistant. By introducing unwanted 1 bits, an attacker might cause spurious messages at the decoding side and but still cannot jam the signal without being have to spend far more energy than the sender. If the attacker jams all the bits in a section, the receiver can always discover the original codeword in the packet. Because of the jammed bits however, the decoding process will generate spurious messages and the receiver can employ digital signatures or other techniques to distinguish genuine messages from spurious messages.

The new HBT algorithm solves the same problem as the

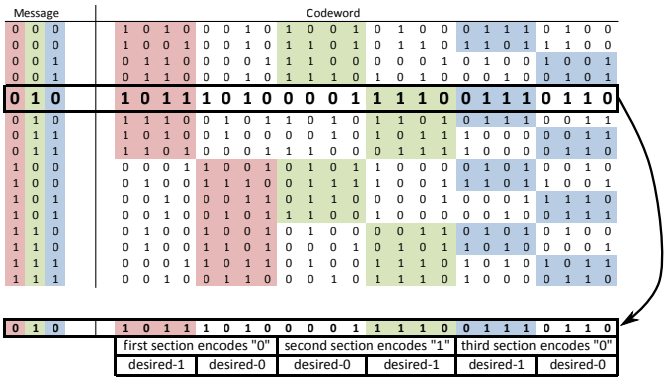


**Figure 4: HBT codeword decoding: converting a given codeword to the corresponding message,**

older BBC algorithm, using a very different approach. These HBT codes are much more general, and actually include BBC codes as a special case. In fact, it can be proved that all possible concurrent codes are special cases of HBT codes, as shown in Theorem 2.

**Theorem 1.** *Given any concurrent code $C$, there exists an HBT code with exactly the same set of codewords, for which packets can be decoded with high probability in polynomial expected time and space.*

*Proof.* If $C$ is any concurrent code (e.g., BBC, or perhaps some concurrent code that will be discovered in the future), then there must exist (by the definition of "concurrent code") an algorithm to find all the codewords in a packet with high probability, in polynomial expected time and space. Define a function $f$ that, given such a packet, uses that algorithm to find any codewords that are in a packet, and returns 1 if there are any codewords, or 0 if there are none. The function $f$ is monotone, because changing a 0 to a 1 in the packet can only add codewords, never delete any existing ones. And it gives the correct answer with high probability in polynomial expected time and space, because the underlying code is a concurrent code. The minimal true vectors for $f$ will be exactly the codewords of the underlying concurrent code, and will also be exactly the codewords for the HBT algorithm using $f$. Therefore, the given concurrent code is also an HBT code. And so, all concurrent codes are special cases of HBT. □

## 8. RESULTS

Tests were run for a large number of circuits. Each circuit had $n$ inputs, feeding into a layer of $n$ AND gates, whose outputs fed into a layer of $n$ OR gates, whose outputs fed into a single AND gate, whose output was the output of the circuit. The layers were not fully connected. Each input sent signals to only $k_1$ of the first-layer AND gates, rather than connecting to all $n$ of them. And each first-layer AND gate received signals from exactly $k_1$ of the inputs. Similarly, each first-layer AND gate's output sent signals to $k_2$ of the second-layer OR gates, and each of those OR gates received signals from $k_2$ first-layer AND gates. The third layer was a single AND gate, receiving inputs from all $n$ of the OR gates.

For each value of $n$, there were many possible choices for the pair $(k_1, k_2)$. For each choice, many circuits were randomly generated, and the average density $a$ was measured. This was measured by randomly generating many circuits of that size and shape, and averaging the density of several random codewords from each circuit. A pair $(k_1, k_2)$ was considered *on the border* if it gave $a \leq 1/9$, and if it was possible to make $a > 1/9$ by adding or subtracting 1 from either $k_1$ or $k_2$.

For each pair on the border, the rate of spurious decodings, $s$, was measured. This was done by generating three random codewords, combining them with a bitwise OR, then decoding the result 50 times, and counting how many distinct codewords were found beyond the original three. For each choice of $n$, the $(k_1, k_2)$ pair with the lowest $s$ was considered to be the best.

Figure 5 shows the best $(k_1, k_2)$ for each input size that is a power of two, from $n = 64$ to $n = 4096$. In each case, the measured $a$, $d$, and $s$, is shown, along with the log of the codebook size $N$, as estimated by the equation in theorem 1. Below the table are five equations for estimating those values, which were found by fitting functions to the numbers in the table. It appears that these give fairly good predictions, especially for the larger circuits, and so might be expected to be roughly correct for even larger circuits.

These results suggest that for this family of circuits, the best circuit is one where each AND gate in the first layer receives input from about 5% of the inputs, and each OR gate in the second layer receives inputs from all but about a square root of those AND gates. This size then gives an average density of 1/9, and a constant number of spurious decodings (about 8). In a practical system, these spurious decodings would be rejected by either using a checksum or a digital signature in each message sent. It is surprising that the number is constant, regardless of the size of the circuit and the size of the resulting codebook. This suggests that a large system would be able to reject the spurious messages very quickly.

The log of the estimated codebook size, $log_2(N)$, grows logarithmically in the number of circuit inputs $n$. The number of message bits that could be sent reliably was found to be just under this number, so the message sizes that could be transmitted also grew logarithmically. The message size grows linearly for BBC, so these initial circuits shown here are not yet as efficient as BBC. A useful area for further research will be to explore other circuits, to find circuit families that will be as efficient as BBC.

# 9. CONCLUSIONS

We proposed the HBT algorithm, a new algorithm for jam resistance without shared secrets, for adversaries with no bound on computational power. This is only the second known algorithm for this problem, after BBC, and uses a very different approach. We proved that it is more general, including BBC as a special case, and even including all possible concurrent codes as special cases. The most difficult part of developing HBT was finding monotone boolean functions (MBF) that would have the desired properties. We described a family of such circuits, and gave both theoretical and empirical results suggesting that these are useful. Areas for future research include how to improve the choice of MBF further, to increase the efficiency, so the message size would grow faster with increasing input size. For this, it may be useful to explore circuits with more layers, or with connections between non-adjacent layers, or with other topologies.

| $n$ | $k_1$ | $k_2$ | $a$ | $d$ | $s$ | $log_2(N)$ |
|---|---|---|---|---|---|---|
| 64 | 3 | 57 | 0.0922 | 0.333 | 7.886 | 6.6 |
| 128 | 7 | 119 | 0.1081 | 0.564 | 7.529 | 8.1 |
| 256 | 14 | 243 | 0.1097 | 0.716 | 7.933 | 9.6 |
| 512 | 28 | 495 | 0.1097 | 0.825 | 8.041 | 11.1 |
| 1024 | 56 | 999 | 0.1101 | 0.898 | 7.704 | 12.5 |
| 2048 | 113 | 2015 | 0.1103 | 0.942 | 7.767 | 13.9 |
| 4096 | 223 | 4044 | 0.1107 | 0.967 | 7.767 | 15.6 |

$$
\begin{array}{ll}
k_1 & = 0.545n \\
k_2 & = n - n^{0.463} \\
a & = 1/9 \\
s & = 8 \\
log_2(N) & = 1.47 log_2(n) - 2.18
\end{array}
$$

**Figure 5: Best results for circuits of various sizes**

# 10. ACKNOWLEDGMENTS

This work is funded in part by the National Science Foundation under Grant No. DUE-0911991. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect those of the National Science Foundation.

This work was sponsored in part by the Air Force Office of Scientific Research (AFOSR). This material is based on research sponsored by the United States Air Force Academy under agreement number FA7000-14-2-0009. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the United States Air Force Academy or the U.S. Government.

# 11. REFERENCES

[1] Leemon C. III Baird and William L. Bahn. An efficient correlator for implementations of bbc jam resistance. Technical Report USAFA-TR-2009-ACCR-02, U. S. Air Force Academy, Academy Center for Cyberspace Research, Nov 2009.

[2] Leemon C. III Baird, William L. Bahn, Michael D. Collins, Martin C. Carlisle, and Sean Butler. Keyless jam resistance. In *Proceedings of the 8th Annual IEEE SMC Information Assurance Workshop (IAW)*, pages 143–150, June 20-22 2007.

[3] Jack P.F. Glas. On multiple access interference in a ds/ffh spread spectrum communication system. In *In the proc of the Third IEEE International Symposiumon Spread Spectrum Techniquesand Applications*, July 1994.

[4] Leemon C. Baird III, William L. Bahn, and Michael D. Collins. Jam-resistant communication without shared secrets through the use of concurrent codes. *U.S. Air Force Academy Technical Report*, February 2007.

[5] E. G. Kanterakis. A novel technique for narrowband/broadband interference excision in ds-ss

communications. In *MILCOM '94*, volume 2, pages 628–632, 1994.

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

[7] L. Li and L. Milstein. Rejection of pulsed cw interference in pn spread-spectrum systemsusing complex adaptive. In *IEEE Trans. Commun*, 1994.