Exhaustive Attack Analysis of BBC With Glowworm for Unkeyed Jam Resistance

Leemon C. Baird III
Email: Leemon@Leemon.com

Bill Parks
Academy Center for Cyberspace Research
US Air Force Academy

***Abstract*—An important problem is to provide jam resistance for wireless networks. This is more difficult in cases such as GPS, where the sender and receiver do not have a shared secret that is unknown to the jammer. Currently, the only known system for such jam resistance without shared secrets is the BBC algorithm, which is fastest when it uses the Glowworm hash. We present a new type of analysis of Glowworm, using exhaustive search to find the absolutely best possible attack for reduced forms of Glowworm, as well as for the full Glowworm applied to shorter packets. Because we are defending against a stronger threat model (an adversary with infinite computational power), the analysis is actually easier than for a traditional threat model, and we derive much stronger results than would be possible for cryptographic hashes that are designed for a more traditional use. In addition, surprising results were found for its behavior near the boundary conditions. The result is that BBC with Glowworm can be used with great confidence, and it is now clear how to choose the best combination of parameters for practical use.** [1]

I. Introduction

It is an important problem to provide jam resistance for wireless networks, especially in the case where they are large enough that it is inconvenient or impossible to have a shared secret known to the sender and receiver, but unknown to the attacker. An extreme example would be GPS satellites, where a secret cannot be shared with the receiver and kept secret from the adversary, because the "receiver" is every person on earth, including the adversaries. There is currently only one algorithm for achieving jam resistance with no shared secret in the presence of an adversary assumed to have infinite computational power: the BBC (Baird, Bahn, Collins) concurrent code [1], [2], [3], [4], [5], [6], [7], [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18].

The BBC jam resistance system is built on an *OR channel*, which is a way of communicating bits, where it is easy for an attacker to change a 0 to a 1, but hard to change a 1 to a 0. In [13], it is shown how this can be built using a Golay efficient Golay correlator, as shown in figure 1.

For fixed delays D_i and weights w_i, this construction allows the generation of a *chip sequence*, a sequence of +1 and -1 values, that is very long: exponential in the number of stages. It is generated by sending into the input at the left end a sequence of 0 values, with a single 1 bit among them. The output starts when the 1 bit is received, and generates the sequence. If several of the input bits are 1, then it generates the sum of several copies of the sequence, each starting when the 1 is received, as if there were several generators whose outputs were being added. By negating the weights, this generates a matched filter. If the generated sequence is fed into the matched filter, its output is the original sequence of 0 and 1 bits. If there is noise in the transmission, then the output will be low, with spikes at each 1. An attacker can easily add new 1 bits (or spikes) to the receiver's output by simply broadcasting the same chip sequence. But it is difficult for an attacker to remove a spike. To do so would involve somehow canceling a complicated, spread-spectrum signal, which is difficult. Therefore, the construction in the figure can be used to create an OR channel.

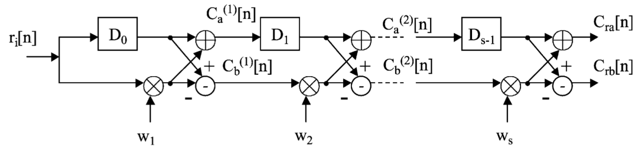


Fig. 1. An efficient Golay correlator. Given a stream of bits as inputs, it outputs the result of summing several Golay chip sequences, one for each 1 bit in the input, each starting at the time the 1 was received. Similarly, by negating the weights, it acts as a matched filter, taking in that waveform, and outputting a low signal with a spike for each 1. We refer to the binary sequence of inputs or output spikes as a "packet".

II. BBC

The BBC algorithm then sends a message by first encoding it as a *packet*, which is a long sequence of mostly 0 bits, with a few 1 bits in it. It is designed to be easy for the receiver to recover the message from the received packet, even if an attacker has added a number of 1 bits. Generally, it only fails if the attacker can fill more than a third of this packet with 1 bits. Normally, the packet would be very sparse, and the sender would only expend energy proportional to the number of 1 bits in the packet. So if the sender is filling, for example, only one position out of 3000 with a 1 bit, then the attacker would need to fill 1 of every 3, which would require 1000 times the broadcast energy. That is why the system is resistant to jamming.

[1]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.

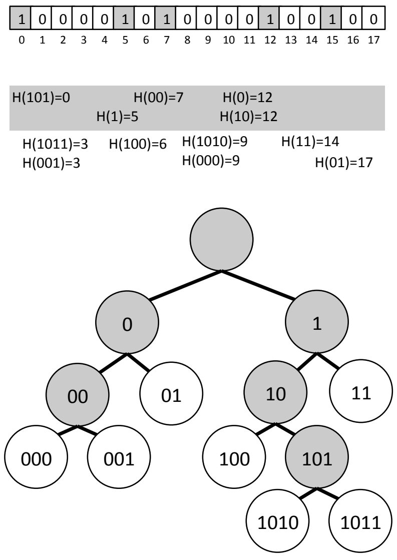
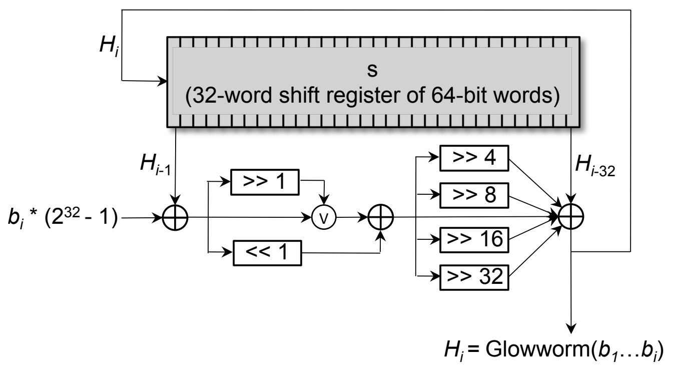


Fig. 2. Use of a hash in BBC decoding. The receiver detects the packet at the top, and decodes it by a depth-first search of the tree at the bottom, using the hash values shown in the middle. A string in the tree is defined to be a non-leaf (gray) if its hash returns the index of a position in the packet containing a mark (a gray 1).



H_i
s
(32-word shift register of 64-bit words)
H_{i-1}
H_{i-32}
$b_i * (2^{32} - 1)$
>> 1
<< 1
v
>> 4
>> 8
>> 16
>> 32
H_i = Glowworm($b_1 \ldots b_i$)

Fig. 3. The Glowworm hash (also called Glowworm-32). The hash H_i of an i-bit string is calculated by XORing the least significant 32 bits of H_{i-1} with the last bit of the string, b_i, then performing a nonlinear transformation on it, then XORing the result with the earlier hash H_{i-32}. Only the lowest 32 bits of H_i are intended to be used as the hash, though the new results found here suggest it is better to use only the lowest 30 bits. Glowworm-N is the same, except each 32 is replaced with N, the word size is $2N$, and the final shifts are by $\{N, N/2, N/4, ...\}$, while greater than 3.

However, there is an attack to consider: we might imagine that a clever attacker could construct a packet that is still sparse, but that somehow forces the receiver into doing too many computations to decode it. This is called an *attack packet*. We might wonder whether an attacker with infinite computational power could find such attack packets, and use them to jam the receiver by overwhelming the receiver's limited computational ability. For some hash functions, such attack packets exist: there are sparse packets that require many operations to decode. For an ideal hash function, such attack packets would not exist: every sparse packet would require only a small number of operations to decode. The purpose of this paper is to explore whether such attack packets exist for BBC using Glowworm. It will be seen that they do not, at least for the small cases that can be exhaustively explored. Thus, for these cases, an attacker with infinite computational ability will not be able to find an attack packet (because they don't exist), and so will not be able to jam without resorting to using large amounts of RF energy.

To understand how an attack packet might work, it is useful to see how BBC decodes a packet to get a message (or multiple messages, if several had been sent simultaneously). The BBC signal is transmitted with no shared secret. The receiver then decodes it using the process in Fig. 2. At the top of the figure is the packet that the receiver receives. The receiver is assumed to be using a hash function H, which is also used by the sender, and is also known to the attacker. The receiver then imagines an infinite binary tree, a portion of which is shown in the figure. Each vertex in the tree contains a binary string. Its children contain the same string, except a 0 is appended to it by the left child, and a 1 by the right child. The receiver walks down this tree, doing a depth-first search, cutting off the search whenever a white vertex is reached. The vertices are colored gray or white according the hash of their string. The output of the hash is viewed as the address of a bit in the packet. The vertex is gray if the bit is 1, and white if it is 0.

Usually, if the packet is less than one third filled with 1 bits, then the gray vertices in the tree will be less than twice as numerous as the 1 bits in the packet. So the time to decode the tree is linear in the number of 1 bits, which are at most linear in the size of the packet. But for certain, weak hash functions, it turns out that many vertices in the tree map to the same location. If that location has a 1, then all those vertices become gray. If the gray vertices form a large, connected set, including the root, then the receiver will forced to explore an enormous set of vertices, and the jammer will have succeeded.

III. Glowworm and Glowworm-N

Of course, the security of this process depends on the nature of the hash. A typical cryptographic hash should be secure, but is very slow. The Glowworm hash was proposed to be fast, and secure when used in BBC. It was not designed to meet the security requirements of a cryptographic hash, such as *collision resistance*, (which means it is hard to find two strings that hash to the same value). Glowworm was designed to meet the security requirements for use within BBC. To be secure, it would need to prevent good attack packets from existing. The empirical results below suggest that Glowworm does achieve this goal. The system is secure against an opponent with infinite computational power, but limited RF broadcast energy. At least for the small cases tested exhaustively here.

Figure 3 shows the standard Glowworm algorithm, also known as Glowworm-32, which was published in MILCOM-2012. The results below will show that it appears to be secure, at least for small packets. But this is not sufficient. It is

```
// Glowworm-N - A hash for use with BBC codes
// 2015, Version 1.0 - this is public domain
// Leemon Baird, Leemon@Leemon.com
//
// Call glowwormInit once. Then repeatedly call AddBit to
// add a new bit to the end of the string so far, and return
// the hash of the resulting string.  DelBit deletes the last
// bit, and returns the hash of the resulting string.
// DelBit and must be passed the last bit of the most recent
// string hashed. The macros should be passed these:
//      uint64 s[N];     //buffer
//      static uint64 n; //current string length
//      uint64 t, i, h;  //temporary
//      const uint64 CHECKVALUE = 0xCCA4220FC78D45E0;
// The integer type must be unsigned with at least 2*N bits.
// For a weakened, reduced form of Glowworm, define N<32,
// and use only the N-2 lowest bits from the output.  For
// N=32, this is normal Glowworm, and 30 bits may be used.
// The third line of AddBit is shown with terms
// from 0 to 3, but for N>32 should have terms ranging
// from 0 to at least log2(N)-3. Init returns the hash of
// the empty string, which for N==32 should be CHECKVALUE.

#define N 32

#define f(t,S) (((N>>S)<4) ? 0 : t>>(N>>S))

#define glowwormAddBit(b,s,n,t) (            \
    t  = s[n % N] ^ ((b) ? ((1L<<H)-1) : 0), \
    t  = (t|(t>>1)) ^ (t<<1),                \
    t ^= f(t,0) ^ f(t,1) ^ f(t,2) ^ f(t,3),  \
    n++,                                     \
    s[n % N] ^= t                            \
    )

#define glowwormDelBit(b,s,n,t) (            \
    n--,                                     \
    glowwormAddBit(b,s,n,t),                 \
    n--,                                     \
    s[n % N]                                 \
)

#define glowwormInit(s,n,t,i,h) {            \
    h = 1;                                   \
    n = 0;                                   \
    for (i=0; i<H; i++)                      \
        s[i]=0;                              \
    for (i=0; i<128*H; i++)                  \
        h=glowwormAddBit(h & 1L,s,n,t);      \
    s[n = 0]                                 \
}
```

Fig. 4. C implementation of the Glowworm-N hash.

common to test cryptographic systems by attempting to break smaller, or weakened versions of them. Therefore, we now propose Glowworm-N, which is the same as Glowworm when N is 32, but which has a much smaller internal state when N is smaller. An implementation in C is given in figure 4.

IV. Testing Full Glowworm, Small Packets

Figure 5 shows the results of exhaustive search for attack packets for the full Glowworm-32, with various small packet sizes. For a given packet size, we exhaustively searched through all possible packets of that size that were one third filled with 1 bits. There are exponentially many, of course, but we were able to search up to packets of size 42. Only packet sizes that are a multiple of 3 were considered, because otherwise they wouldn't be exactly one third filled with 1 bits. Figure 6 shows the same data, but with each cost divided by the packet size, to give a ratio of cost per packet bit.

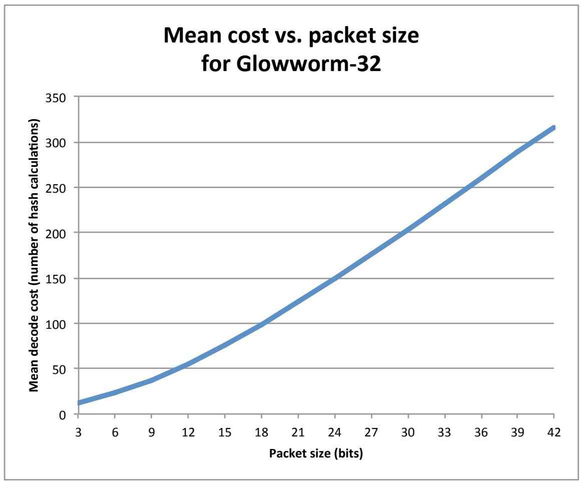


Fig. 5. Average decode cost (number of hashes calculated while decoding) for a worst-case packet, as a function of the packet size, for full Glowworm (Glowworm-32). The graph shows how successful an attacker with infinite computational power would be. It becomes very linear above size 12, suggesting that packets of millions of bits would still easy to decode, even when attacked by such an adversary.

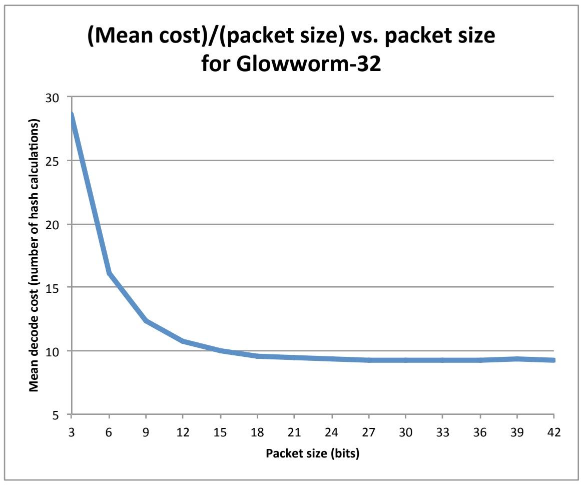


Fig. 6. Average decode cost (number of hashes calculated while decoding) divided by the packet size, for a worst-case packet, as a function of the packet size, for full Glowworm (Glowworm-32). This levels off at 9.29, meaning that on average, the most powerful attack packet possible requires only 9.29 hash calculations per packet bit.

However, it wasn't enough to simply do this with standard Glowworm, which is a single hash. For example, on a packet of size 6, there are few possible packets, and one of them will have the highest decode cost, so that could be considered the worst-case cost for size 6. But it will be highly dependent on the particular way the hash was seeded, and will not be very informative about the hash algorithm itself. Therefore, we used a standard approach from cryptography: we expanded the single hash function into a family of hash functions by using random prefixes. In other words, for a string S, we could define the result of applying hash function Glowworm-32-R to string S to be the result of applying Glowworm-32 to to the string (R,S) which is concatenation of R and S. In this way,

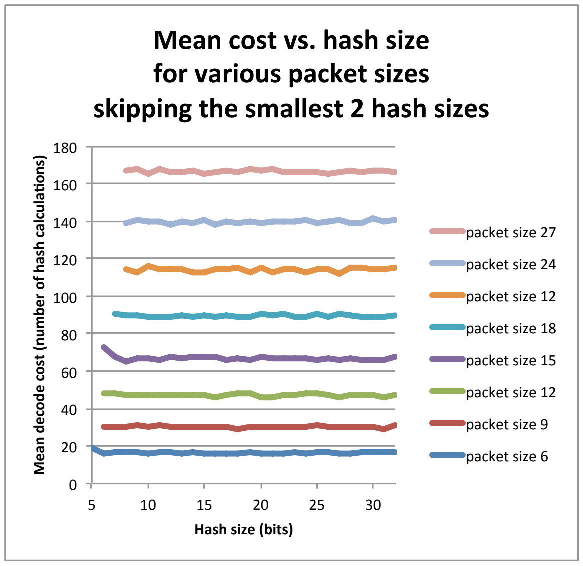


Fig. 7. Average decode cost (number of hashes calculated while decoding) for a worst-case packet, as a function of the packet size, for Glowworm-N, for various sizes of packet and various sizes of N. Packet sizes are always 2^{N-2} or less. The lines are horizontal and flat, indicating that all N values are equally secure. The lines are evenly spaced, because figure 5 is linear, so the system is secure for these cases.

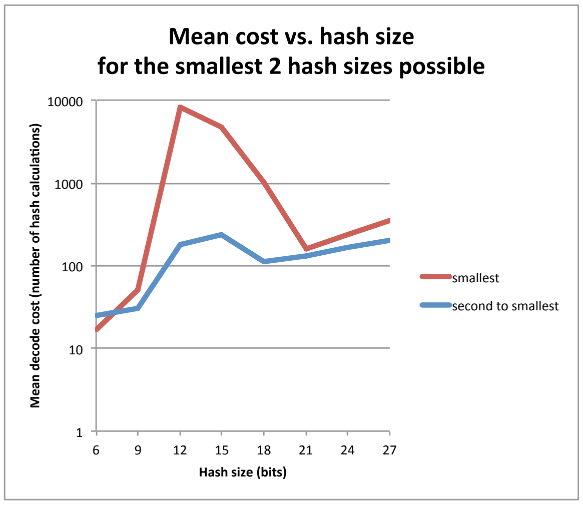


Fig. 8. Average decode cost (number of hashes calculated while decoding) for a worst-case packet, as a function of the packet size, for Glowworm-N, for various packet sizes and two sizes of N. In one curve, the N is chosen for each packet to be as small as possible without the packet being larger than 2^N. The other curve is for the second-smallest N. The costs are high and the curves are nonlinear, indicating that the system is insecure for these 2 lowest N values.

we generated 500 different hash functions in the Glowworm-32-R family for each packet size, and exhaustively searched to find the most powerful attack packet for each of the 500 hashes. Figure 5 shows the average of those 500 trials. The variance was also found to be small, and to actually decrease as a fraction of packet size, as the packet size increased.

The graph in figure 5 appears to be a very straight line after the first two points (at sizes 6 and 9). From 12 on, it seems to be very straight. Therefore, figure 6 appears to level out at just 9.29 hashes calculated per packet bit. If this trend continues for large packets, then it means Glowworm-32 is secure against adversaries with infinitely powerful computers, for packets with millions of bits.

However, it is not obvious that this nice, linear trend would continue. Glowworm-32 has a buffer of 32 words. So when a message is more than 32 bits long, it wraps around and starts to interact with parts of the buffer that have been seen before. Unfortunately, a packet of size 42 will have only 14 bits in it set to 1, which means the tree will typically be explored only to a depth of about 14. It will never reach a depth of 32, so it will never wrap around. This is why we also went on to perform an even larger search through the families of weakened Glowworm: which is Glowworm-N for small N.

V. Testing Reduced Glowworm

For reduced Glowworm, we still did exhaustive searches over all packets of a given density, to find the one that has the highest cost. We also still did that 500 times for each packet size, and averaged the results. And we did all that for many packet sizes. But, in addition, we did all of that for *each* value of N from 5 to 32. This was about 27 times as much computation as the results from the previous section, but the results were worth it.

Figure 7 shows the average cost to decode the most powerful attack packet (vertical axis) versus the value of N for Glowworm-N (horizontal axis), with a separate curve for each packet size. The result is surprising: the curves are all horizontal. This means that in each case, the weakened form of Glowworm was just as strong as the original! This is surprising, because the weakened forms of Glowworm, especially near the left end of each line, are cases where Glowworm wrapped around its tiny buffer many times during decoding. This suggests that Glowworm is inherently strong, and doesn't need the large amount of internal buffer, except when working with very large packets.

We can't guarantee that this behavior will continue all the way up to billion-bit packets and Glowworm-32, but it is certainly encouraging. And we do know, for certain, that for the small cases we tested, an adversary with infinite computational power cannot break the system, because the attack packets that the attacker would be searching for, simply don't exist.

However, the lines in figure 7 are actually incomplete. The leftmost 2 data points on each line have been deleted. That is because those points are actually *not* secure. So it is interesting to analyze them separately.

Figure 8 shows all the results that were left off of the previous figure. For the first time, the vertical scale is now logarithmic, because the costs have exploded exponentially! The meaning here is clear. If you have a packet of, say, 27 bits, then clearly you need at least 5 bits to address each position (because $2^5 = 32$ is big enough, but $2^4 = 16$ is not). Therefore, there is no way to test a 27-bit packet with

Glowworm-4 or Glowworm-3 or Glowworm-2. It needs at least Glowworm-5. Surprisingly, it turns out that the 27-bit packet is completely secure with Glowworm-7 and above. But Glowworm-5 and Glowworm-6 are the smallest two possible values of N, and they turn out to be insecure. In fact, as the figure shows, the smallest two values of N are insecure for all the cases tried.

This suggests that Glowworm-32 would not actually be secure for a packet of 2^{32} bits, nor for a packet of 2^{31} bits. But it would be fine for a packet of 2^{30} bits. At least, if the trends continue. This is a useful result to know. It is unlikely that we will ever need packets of 4 billion or 2 billion bits. One billion certainly sounds sufficient. But if we ever do, we should treat Glowworm-32 as only being sufficient for generating 30-bit hashes. If we ever do need a 32-bit hash, we will have to go to at least Glowworm-34. This would require integers of 68 bits, which is likely to be much slower on current hardware. So this is certainly useful information to have in the future.

VI. Conclusions

In cryptography, a system is only trusted after a number of people have tried to find attacks, and failed. Or if it can be proved to be equivalent to another system where that has happened. Even when researchers attack a reduced form of a cryptographic hash and "break" it, they can never be sure that their attack was the most efficient possible. So results on the strength of cryptographic hashes must always be interpreted cautiously.

The results presented here for BBC with Glowworm are stronger than that. We have shown that, at least for small packets or reduced Glowworm, the system is *definitely* secure against *all possible* computational attacks, even if the attacker has infinite computational power. Because successful attack packets simply don't exist. This result is stronger than a typical result against a cryptographic hash. For traditional hashes, it is already known that collisions must exist, and the only question is whether they can be found efficiently. But in this paper, we have shown that effective attack packets simply don't exist. At least not for those small cases. So this gives us an unusually high degree of confidence in the security of the system.

Of course, for large packets with full Glowworm, it is theoretically possible that some unknown attack can still exist. But such an attack would be somewhat surprising, given the guaranteed non-existence of attacks on smaller versions. Especially since these smaller versions were still large enough so that a message would wrap around the buffer multiple times. Therefore, we consider it very likely that no such attack can exist on the larger system, either.

Two interesting results were found, which should guide the choices of parameters in practical uses of BBC with Glowworm. First, for packets of size 2^k bits, Glowworm-N is secure for $N = k + 2$, and there is no need for (and no additional security from) increasing N beyond that. Second, for $N = k + 1$ and $N = k$, the system is no longer secure. This suggests that standard Glowworm (which is Glowworm-32) should only be used for a 30-bit hash or smaller, not a 32-bit hash. Only its lowest 30 bits of output should be used, rather than the lowest 32 bits. This should still be sufficient for real-world applications, since it would allow packets with over a billion bits. But it is useful to know that if a need ever arises for more packet bits than that, then it will be necessary to use Glowworm-N with an N greater than 32, and so with integer variables of more than 64 bits.

References

[1] L. C. Baird III, W. L. Bahn, and M. D. Collins, "Jam-resistant communication without shared secrets through the use of concurrent codes," U. S. Air Force Academy, Tech. Rep. USAFA-TR-2007-01, Feb 14 2007.

[2] W. L. Bahn, L. C. Baird III, and M. D. Collins, "The use of concurrent codes in computer programming and digital signal processing education," *Journal of Computing Sciences in College*, vol. 23, no. 1, pp. 174–180, Oct 2007, also in the Proceedings of the 16th Annual Rocky Mountain Conference of the Consortium for Computing Sciences in Colleges (RMCCSC), Orem Utah.

[3] W. L. Bahn and L. C. Baird III, "Impediments to systems thinking: Communities separated by a common language," in *Proceedings of the 4th International Conference on Cybernetics and Information (CITSA)*, July 12-15 2007, pp. 122–127.

[4] L. C. Baird III, W. L. Bahn, M. D. Collins, M. C. Carlisle, and S. Butler, "Keyless jam resistance," in *Proceedings of the 8th Annual IEEE SMC Information Assurance Workshop (IAW)*, June 20-22 2007, pp. 143–150.

[5] L. C. Baird III and D. H. Kraft, "A new approach for boolean query processing in text information retreival," in *Proceedings of the International Fuzzy Systems Association (IFSA) 2007 World Congress*, June 18-21 2007.

[6] D. Schweitzer, L. C. Baird III, and W. Bahn, "Visually understanding jam resistant communication," in *Proceedings of the 3rd International Workshop on Visualization for Computer Security*, Oct 29 2007, pp. 175–186.

[7] W. L. Bahn and L. C. Baird III, "Extending critical mark densities in concurrent codecs through the use of interstitial checksum bits," U. S. Air Force Academy, Academy Center for Cyberspace Research, Tech. Rep. USAFA-TR-2008-ACCR-02, Dec 8 2008.

[8] ——, "Hardware-centric implementation considerations for bbc-based concurrent codecs," U. S. Air Force Academy, Academy Center for Cyberspace Research, Tech. Rep. USAFA-TR-2008-ACCR-03, Dec 8 2008.

[9] W. L. Bahn, L. C. Baird III, and M. D. Collins, "Jam resistant communications without shared secrets," in *Proceedings of the 3rd International Conference on Information Warfare and Security (ICIW08)*, April 24-25 2008.

[10] W. L. Bahn, L. C. Baird III, and D. Collins, Michael, "Oscillator mismatch and jitter compensation in concurrent codecs," in *IEEE Military Communication Conference (MILCOM08)*, Nov 17-19 2008.

[11] R. Thurimella and L. C. Baird III, *Cryptography for Cyber Security and Defense: Information Encryption and Cyphering*. IGI Global, 2009, chapter title: "Network Security".

[12] L. C. Baird III and W. L. Bahn, "Parallel bbc decoding with little interprocess communication," U. S. Air Force Academy, Academy Center for Cyberspace Research, Tech. Rep. USAFA-TR-2009-ACCR-01, Nov 2009.

[13] ——, "An efficient correlator for implementations of bbc jam resistance," U. S. Air Force Academy, Academy Center for Cyberspace Research, Tech. Rep. USAFA-TR-2009-ACCR-02, Nov 2009.

[14] ——, "An o(log n) running median or running statistic method, for use with bbc jam resistance," U. S. Air Force Academy, Academy Center for Cyberspace Research, Tech. Rep. USAFA-TR-2009-ACCR-03, Nov 2009.

[15] S. Hamilton, "Secure jam resistant key transfer," Masters thesis, Auburn Univeristy, Tech. Rep., May 2008.

[16] M. Kuhr, "An adaptive jam-resistant cross-layer protocol for mobile adhoc networks in noisy environments," PhD thesis, Auburn Univeristy, Tech. Rep., May 2009.

[17] D. Sanders, "A single-hop medium access control layer for noisy channels," PhD thesis, Auburn Univeristy, Tech. Rep., August 2009.

[18] S. S. Hamilton and J. A. Hamilton Jr., “A secure jam resistant key transfer : Using the dod cac card to secure a radio link by employing the bbc jam resistant algorithm,” in *IEEE Military Communication Conference (MILCOM08)*, Nov 17-19 2008.