Looping for Encryption Key Generation over the Internet: A New Frontier in Physical Layer Security

Amir K. Khandani

E&CE Dept., Univ. of Waterloo, Waterloo, Ontario, Canada; khandani@uwaterloo.ca

Abstract: Current key sharing techniques rely on the hardness of solving a solvable, but complex, mathematical problem. This entails, in Information Theoretical sense, the encryption key is not secret, it can be found by solving the underlying mathematical problem. Sensitive data we encrypt today using traditional techniques can be recorded by malicious parties and be deciphered in the future whenever improved hacking techniques and supporting computing technology permit. Information Theory proves the existence of methods for sharing of encryption keys that are unconditionally secure, but does not show how to bring such theoretical results to practical use. One of the central information theoretical approaches to key sharing is based on exploiting common randomness. This theoretical result states that if two dependent random variables, A and B, are available at Alice and Bob, then, by communicating through a public channel between A and B, it is possible to securely share a key of size $I(A;B)$. To bridge the gap between theory and practice, one needs a method to generate two sets of dependent random variables, one at Alice's side and the other one at the Bob' side, as well as a method to extract two identical keys from these dependent random variables. This article presents a novel technique to achieve this goal over the Internet. Dependent random variables are generated by measuring packet travel times between Alice and Bob, and error-free key extraction from dependent random variables is realized by using a randomized Low Density Parity Check Code (LDPC). Through looping of packets between Alice and Bob, the mutual information between random variables is increased. Finally, methods are presented to measure the likelihood values required in decoding the underlying LDPC. It is shown that the key rate is approximately equal to $0.5\log_2\left(4L^2/(4L-1)\right) \approx 0.5\log_2(L)$ where L is the number of round trips (loops). Test results (based on measurements between distant nodes over the Internet) are presented, demonstrating the feasibility of the proposed technique. The proposed method is implemented entirely in software (through high-level programming, e.g., using C-language, at the application layer). This operation does not require modifying the underlying network.

*Author gratefully acknowledges Neda Mohammadizadeh for her contribution in gathering reported simulation results related to actual measurements over Azure Microsoft Cloud.

1- Introduction: Key sharing involves generating two identical strings of binary numbers at two different locations without disclosing any information to potential eavesdroppers. Typically, keys of size 256 bits are needed to be used with Advanced Encryption System (AES) to encrypt data for secure "transmission", "sharing" or "storage". Ideally, encryption keys should be truly random, can be shared between legitimate parties without disclosing any information about the key bits, and can be renewed whenever desired. Techniques presented in this article address this problem.

2- Proposed Method: To establish a key between a Node A (Alice) and a Node B (Bob) over the Internet, Node A sends a sequence of N User Datagram Protocol (UDP) packets, indexed from 1 to N, at regular time intervals (typically with 10 msec time gap) to Node B. Each UDP packet contains its corresponding index, and (optionally) a set of random bits. Indices of packets are used to generate the key pair, and if available/needed, packets' contents can be used to enlarge the key size. Node B sends the received packets, one by one, back to Node A and Node A sends them back to Node B. Upon L rounds of looping (called multi-loop), Node A and Node B separately measure their corresponding total round trip times for each packet. Figure 1 shows an example for $L = 2$ loops (i.e., two complete loops plus a single hop to close the final loop). Due to looping, these round-trip times, although random, will be close to each other (will have $2L - 2$ common travel times). This is due to the fact that time is additive in successive looping of a given packet. Figure 2 shows an example for $L = 2$.

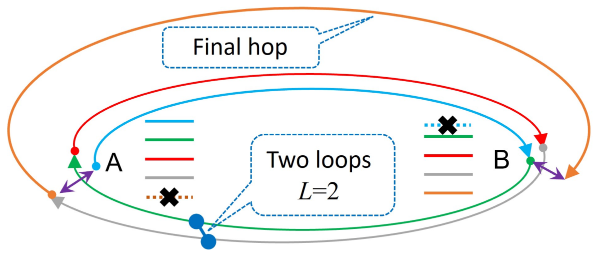


Figure 1: Round trip times measured at Node A (Alice) and at Node B (Bob) for two loops (two complete loops plus a single hop to close the final loop).

Delay values corresponding to passing the packets from the receiver front-end to the transmitter front-end (within Node A or Node B), i.e., ϵ_1, ϵ_2, ϵ_3, ϵ_4 in Fig. 2, are small compared to travel times across network links. If the number of loops is equal to L, then there will be $2L$ such ϵ values which, although small, are *all in common* between

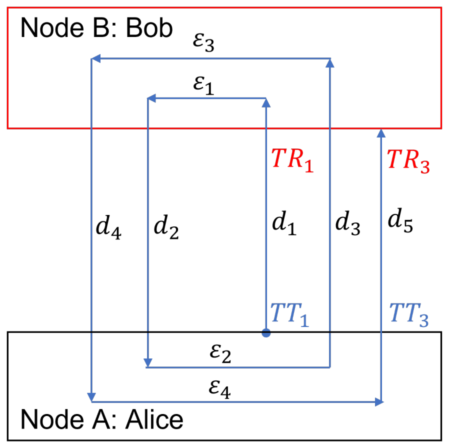
$$T_A = TT_{L+1} - TT_1 \quad \text{and} \quad T_B = TR_{L+1} - TR_1 \quad (1)$$

where T_A and T_B denote the round trip times for Node A and Node B, respectively . For simplicity of notation, ϵ values are dropped from expressions hereafter. Let us use the notations

$$T_A^{(\ell)} = TT_{L+1}^{(\ell)} - TT_1^{(\ell)} \quad \text{and} \quad T_B^{(\ell)} = TR_{L+1}^{(\ell)} - TR_1^{(\ell)} \quad (2)$$

to denote the total travel times, measured by Node A and Node B, respectively, for packet indexed by ℓ. Note that since each node measures the difference between two time values (relying on each node's local clock), the two nodes do not need to be time synchronized. As an example, let us focus on the multi-loop $A \rightarrow A$. In a configuration with L loops, there will be a total of $2L$ delay values contributing to such a multi-loop, followed by a delay value for the final link from A to B. A similar argument applies to the $B \rightarrow B$ multi-loop (see Figs. 1 and 2).

2.1- Extracting bit values from measured travel times: Each node records its measured multi-loop travel time for each UDP packet, computes the mean of the recorded multi-loop travel times, and then assigns a binary value of zero to packets with a travel time less than the mean value; and a binary value of one to those with a travel time higher than the mean. The resulting bit streams have many common values since the measured multi-loop travel times differ only in two links among the $2L$ links forming each multi-loop, namely the first link and the last link connecting $A \rightarrow B$. Travel times in the first link and the last link form a pair of dependent random variables. Since these two links connect the same pair of nodes in the same direction, it is likely that they pass through the same physical connections/paths in the network. This increases the dependency between respective travel times. On the other hand, since the first and the last transmissions become active at different times, the dependency between the two travel times reduces. In summary, $2L-2$ summands forming



$$TT_3 - TT_1 = \underline{d_1 + \varepsilon_1 + d_2 + \varepsilon_2 + d_3 + \varepsilon_3 + d_4 + \varepsilon_4} + 0$$

$$TR_3 - TR_1 = 0 + \underline{\varepsilon_1 + d_2 + \varepsilon_2 + d_3 + \varepsilon_3 + d_4 + \varepsilon_4} + d_5$$

$$D_c = \varepsilon_1 + d_2 + \varepsilon_2 + d_3 + \varepsilon_3 + d_4 + \varepsilon_4 \approx d_2 + d_3 + d_4$$

Figure 2: Round trip times measured at Node A (Alice) and at Node B (Bob). ϵ_1, ϵ_2, ϵ_3, ϵ_4 denote delay values incurred within Node A and Node B, respectively. Note that seven out of eight components forming $TT_3 - TT_1$ and $TR_3 - TR_1$, namely $\epsilon_1, d_2, \epsilon_2, d_3, \epsilon_3, d_4, \epsilon_4$, are the same in the two measurements. Round trip time measured at Node A and at Node B, namely $TT_3 - TT_1$ and $TR_3 - TR_1$, respectively, are dependent random variables (due to having the value of $\epsilon_1 + d_2 + \epsilon_2 + d_3 + \epsilon_3 + d_4 + \epsilon_4$ in common).

the two multi-loop travel times are identical, and the remaining two are dependent random variables. A higher probabilistic dependency translates to a higher level of common randomness. Next, a method is presented to form/extract likelihood values for each bit.

2.2- Removing discrepancies between bit streams: To remove any remaining discrepancy, Node B forms a randomized parity check generator for an $(n = k + p, k)$ Low-Density Parity Check Code (LDPC) where n is the coded block length, k is the size of data to be encoded and p is the size of parities. Data to be encoded corresponds to the bit stream at Node A, with some error detection bits $\mathbf{d}$ added for error detection. This matrix, denoted as $\mathbf{H}$, is further randomized (by/at Node B) using a procedure similar to what is used in McEliece Cryptosystem [3] (also see [4] and references therein), i.e., Node B computes,

$$\hat{\mathbf{G}} = \mathbf{AHP}. \tag{3}$$

In 3, $\mathbf{P}$ is a permutation matrix (or operator, e.g., Knuth permutation) of size $k \times k$, $\mathbf{H}$ is the $p \times k$ parity check matrix for the LDPC code of length $n = k + p$ and information rate k, $\mathbf{A}$ is an invertable binary matrix of size $p \times p$. Then, Node B sends $\hat{\mathbf{G}}$ over a public channel to Node A. Node A multiplies $\hat{\mathbf{G}}$ by its local copy of the bit stream (composed of bits extracted from multi-loop travel times plus error detection bits), and sends the resulting syndrom, denoted as $\mathbf{s}$, to Node B. Knowing the matrices $\mathbf{A}$, $\mathbf{G}$ and $\mathbf{P}$, Node B will be able to revert the randomness and decode the original LDPC code (locate erroneous bit positions). This is achieved by first computing,

$$\hat{\mathbf{s}} = \mathbf{A}^{-1}\mathbf{s}. \tag{4}$$

It then uses $\hat{\mathbf{s}}$ as the syndrom bits produced by the parity check matrix $\mathbf{HP}$. Note that $\mathbf{HP}$ has a sparse structure similar to the one used by Node B in generating $\mathbf{H}$. It is indeed obtained from $\mathbf{H}$ by permuting the columns of $\mathbf{H}$ according to the permutation $\mathbf{P}$. Node B uses the received $\mathbf{s}$ to remove discrepancies between its local bit stream and that of the Node A (used to generate $\mathbf{s}$). To decode the LDPC code, relying on probability propagation, Node B flips bits in its local bit stream to satisfy parity equations captured in $\mathbf{HP}$ (with right hand side values of the parity equations extracted from $\mathbf{s}$). Once the decoding is completed, the resulting bit stream is permuted by multiplying it with $\mathbf{P}^{-1}$. Then, the added error detection bits are checked to verify if the recovered bits are received free of error (is the same as the one at Node A). If error detection indicates an error, Node B signals Node A to repeat the entire key generation procedure. If error detection passes, Node B signals Node A that the operation has been successful. Upon successful removal of discrepancies, the resulting vector of binary values is used as the encryption key.

2.3- Formation of bit likelihoods: Consider a packet indexed by i and assume its corresponding total travel time at Node B, i.e., $T_B^{(i)}$ defined in 2 is at a distance of $\mathsf{d}_B^{(i)}$ from the average delay at Node B. The corresponding travel time at Node A is equal to $T_A^{(i)}$. Without loss of generality, let us assume $\mathsf{d}_B^{(i)} > 0$, corresponding to a bit value of one (in hard decision decoding). If the average total travel times at Node A and Node B were exactly the same[1], the chances that the corresponding bit to be a zero at Node A would be equal to,

$$P\left(T_A^{(i)} - T_B^{(i)} < \mathsf{d}_A^{(i)}\right) \Longrightarrow P\left(d_1^{(i)} - d_{2L+1}^{(i)} < \mathsf{d}_B^{(i)}\right), \tag{5}$$

where $d_1^{(i)}$ and $d_{2L+1}^{(i)}$ denote the travel times in the first link and the last link for packet indexed by i. We have concluded, through measurements over nodes distributed over Microsoft Azure cloud network, that the probability density function of

$$z = T_A^{(i)} - T_B^{(i)} = d_1^{(i)} - d_{2L+1}^{(i)}, \tag{6}$$

is very close to a Laplace density function, i.e.,

$$f(z) = \frac{\lambda}{2}\exp\left(-\lambda|z|\right) \Longrightarrow \mathrm{var}(z) = E\left(z^2\right) = \frac{2}{\lambda^2}. \tag{7}$$

Noting symmetry, the probability density function of

$$T_B^{(i)} - T_A^{(i)} = d_{2L+1}^{(i)} - d_1^{(i)}, \tag{8}$$

would be the same as the one given in 7. Consequently, the likelihood value of each bit can be obtained, in closed form, by integrating the probability density function in 7. Relevant ranges for integration (at Node B) are obtained from 5. This results in a simple closed form expression for each likelihood value as a function of $\mathsf{d}_B^{(i)} > 0$ and λ. It remains to compute the parameter λ in 7. This is achieved by computing the corresponding variance,

[1]This assumption may adversely affect the final error rate, which, through actual measurements, is shown to be small.

i.e., $E(z^2) = 2/\lambda^2$. On the other hand, noting 8 and assuming random variables $d_{2L+1}^{(i)}$ and $d_1^{(i)}$ are independent[2], it follows

$$E\left(z^2\right) = \frac{2}{\lambda^2} = 2E\left[\left(d_1^{(i)}\right)^2\right]. \tag{9}$$

It remains to compute the variance in 9 at Node B. To do this, we rely on the fact the sequence of UDP packets are transmitted, with equal time gaps, over the first link. Node B measures the received times, $TR_1^{(i)}$, for $i = 1, \ldots, N$ where N is the number of UDP packets. These times are denoted as t_i for packets indexed by $i = 1, \ldots, N$. Due to variation in travel times of successive packets t_i, $i = 1, \ldots, N$ are not uniformly spaced (are not at equal time intervals). We rely on (minimizing) mean square error criterion to shift (regularize) the values of t_i, $i = 1, \ldots, N$ to have equal time gaps. Due to a potential difference between clock rates at Node A and Node B, the time gap at Node B (using its local oscillator as the reference of time) is not necessarily equal to the time gap at Node A. Let us assume the time gap at Node B (if all packets were received in equal time intervals) would be equal to g (measured based on the clock at Node B). To regularize the packets (order them at regular time intervals) at Node B, we rely on the solution to the following problem:

$$\min_{\mathsf{o},\mathsf{g}} \sum_{i=1}^{N} \left(\mathsf{t}_i - \mathsf{o} - i\mathsf{g}\right)^2 \tag{10}$$

where o is an offset and g denotes the regularized time gap at Node B. Setting the derivatives of 10 with respect to o and g equal to zero, one can compute o and g in closed forms. Adjusting (regularizing) the values of t_i, results in

$$\hat{\mathsf{t}}_i = \mathsf{o} + i\mathsf{g}. \tag{11}$$

Finally, we have,

$$E\left[\left(d_1^{(i)}\right)^2\right] \approx \frac{1}{N} \sum_{i=1}^{N} \left(\mathsf{t}_i - \hat{\mathsf{t}}_i\right)^2. \tag{12}$$

Note that one could use the values of t_i in conjunction with their regularized values $\hat{\mathsf{t}}_i$ to compute higher order moments for the desired probability density function and improve the accuracy of $f(z)$ in 7.

[2]This assumption may adversely affect the final error rate, which, through actual measurements, is shown to be small.

3- Independence of Extracted Bits: A requirement in key generation is that the key bits should be independent of each other. This section studies this property in conjunction with the key generation method proposed in this article. Let us use the generic notation T_A and T_B to refer to the random variables in 5 and 8, respectively. Dropping ϵ values from expressions, for L loops, the common random variable between T_A and T_B is equal to

$$D_c = \sum_{i=2}^{2L} d_i \tag{13}$$

where d_i is the random variable corresponding to *variations* in travel time across a single link between Node A and Node B. Note that the relevant average values are subtracted prior to extracting d_i values, and hence d_i values are small random variations around the corresponding statistical means. Random variations are, for example, due to randomness in network queuing time; random variations in network switching time, and/or random variations in extracting packets' headers for network route selection. If random time variations corresponding to successive packets are independent, then the extracted bits will be independent of each other. This means, if the Internet channel, with random time variations considered as channel imperfections, is memory-less, then the extracted bits will be independent of each other. Events such as increase in the size of queued data, and/or buffer overflow, result in lasting effects, and memory in the underlying channel, which may interfere with independence property. It follows that

Increasing the time gap between successive packets reduces channel memory, eventually resulting in a memory-less channel where extracted bits become independent of each other.

Privacy amplification is a technique commonly used to combat any remaining dependencies [6].

4- Computing the Rate of Shared Secret: This section computes the mutual information (rate of shared secret) between Node A and Node B. We have,

$$T_A = D_c + d_1 = \sum_{i=1}^{2L} d_i \tag{14}$$

$$T_B = D_c + d_{2L+1} = \sum_{i=2}^{2L+1} d_i. \tag{15}$$

It is assumed that variations in travel times are independent and identically distributed. As T_A and T_B in 14 and 15 are composed of the summation of $2L$ independent and identically distributed random variables, the marginal probability density functions for T_A and T_B approaches a Gaussian density for large values of $2L$. Using the notation σ^2 to represent

$$\sigma^2 = E\left[\left(d_1^{(i)}\right)^2\right] \tag{16}$$

the variance of T_A and T_B will be equal to

$$E(T_A^2) = E(T_B^2) = 2L\sigma^2. \tag{17}$$

Even though D_c, T_A and T_B tend to Gaussian distributions for their marginals, one cannot conclude that T_A and T_B are jointly Gaussian. Approximating this joint distribution to be a bi-variate Gaussian, one can compute the key rate as

$$\begin{aligned} I(T_A;T_B) &= H(T_A) + H(T_B) - H(T_A,T_B) \quad (18)\\ &\approx \log_2(4\pi e L\sigma^2) - \log_2\left(2\pi e|\mathbf{C}|^{1/2}\right) \end{aligned}$$

$$\mathbf{C} = \begin{bmatrix} 2L\sigma^2 & 2L\sigma^2 - \sigma^2 \\ 2L\sigma^2 - \sigma^2 & 2L\sigma^2 \end{bmatrix} \tag{19}$$

$$|\mathbf{C}| = (4L-1)\sigma^4. \tag{20}$$

Replacing in 18, we obtain

$$I(T_A;T_B) \approx 0.5\log_2\left(\frac{4L^2}{4L-1}\right) \approx 0.5\log_2(L). \tag{21}$$

Figure 3 shows the rate of shared secret. Assuming a Laplace probability density function[3] for delay variations in each link (see expression 16 in [5] with service rate and arrival rate assumed to be equal to each other, i.e., $\lambda = \mu$), the probability density function for T_A and T_B will be the sum of $2L$ i.i.d. Laplace random variables. Figure 3 includes the result of $I(T_A;T_B)$ based on such a Laplace density function for delay variations in each link ($I(T_A;T_B)$ is computed through numerical simulation). Noting that: (i) a Laplace random variable corresponds to the difference between two i.i.d. exponential random variables, and (ii) the sum of i.i.d. exponential random variables results in a Gamma random variable, it is concluded that the probability density functions of D_c, T_A and T_B, each correspond to the difference between two i.i.d. Gamma random variables. These properties can be used to compute the rate of shared secret, but the resulting expressions are complex and such a derivation is beyond the scope of this work. Figure 4 shows a sample of the measured histogram based on measuring $T_B^{(i)} - T_A^{(i)}$) and density function obtained analytically from expressions 7,9,12.

[3]This assumption affects the error rate in key generation, which, through actual measurements, is shown to be negligible/acceptable.

5-Numerical Results: Figure 4 shows an example for the histogram of $T_B^{(i)} - T_A^{(i)}$, and the corresponding density function obtained analytically from expressions 7,9,12 (red). Figure 5 shows the corresponding Cumulative Distribution Function (CDF). Performance of the key generation algorithm is studied using nodes (over Azure cloud) in Canada, Europe and East Asia. Table 1 shows the probability that a common (error-free) key is established in a single attempt. Information block of the LDPC is equal to the number of delays mentioned in 1, plus 32 bits for error checking, resulting in 832 bits, and the number of parities is equal to 416. In decoding the LDPC, reliability values for bits corresponding to UDP packets are computed from density function obtained using expressions 7,9,12 and probabilities for 32 error detection bits are set to 0.5 for zero/one. This means, it is assumed error detection bits are independent and identically distributed with equal probabilities for zero and one.

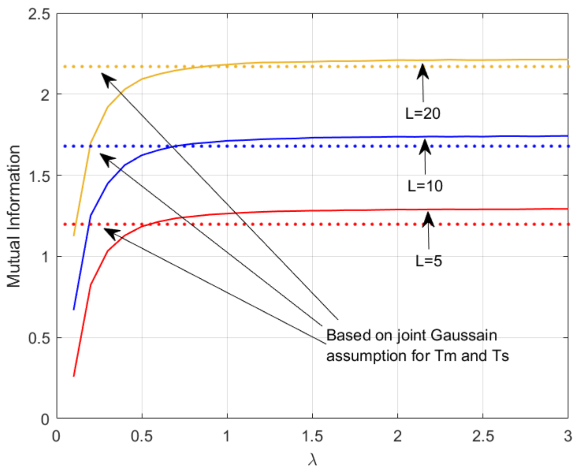


Figure 3: Rate of shared secret, i.e., $I(T_A;T_B)$.

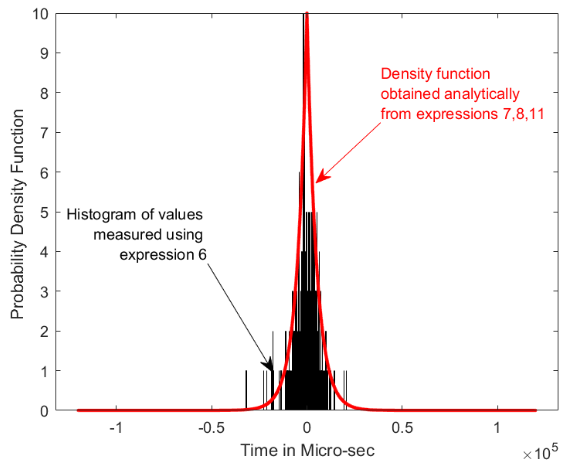


Figure 4: Histogram based on measuring $T_A^{(i)} - T_B^{(i)}$ (black) and density function obtained analytically from expressions 7,9,12 (red).

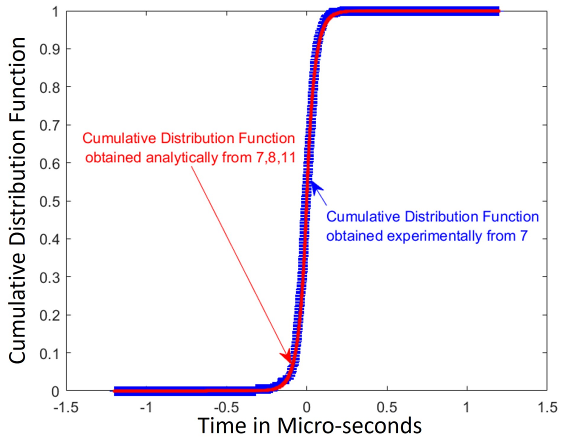


Figure 5: Cumulative Distribution Function (CDF) based on measuring $T_A^{(i)} - T_B^{(i)}$ (blue) and CDF obtained analytically from expressions 7,9,12 (red).

# of Delays	# of Loops	Probability of Success	Location of Node A
800	12	69%	Canada
800	15	85%	Canada
800	20	97%	Canada
800	12	76%	France
800	15	87%	France
800	20	91%	France
800	12	79%	East Asia
800	15	94%	East Asia
800	20	96%	East Asia

Table 1: Probability of success in forming a shared secret in a single round. Node B is within the city of Waterloo, ON, Canada. In all cases, the size of bits used in error checking is 32.

References

[1] US patent 11,057,204, “Methods for encrypted data communications”, A. Khandani (granted)

[2] R. Ahlswede and I. Csiszar, “Common randomness in information theory and cryptography. I. Secret sharing," in *IEEE Trans. on Info Theory*, vol. 39, no. 4, pp. 1121-1132, July 1993

[3] R. J. McEliece, “A public-key cryptosystem based on algebraic coding theory,” *DSN Progress Report*, pp. 114–116, 1978.

[4] D. Engelbert, R. Overbeck and A. Schmidt “A Summary of McEliece-Type Cryptosystems and their Security”, *Journal of Mathematical Cryptology*, pp. 151—199, May 21, 2007

[5] K. Hammad, A. Moubayed, A. Shami and S. Primak, "Analytical Approximation of Packet Delay Jitter in Simple Queues," in *IEEE Wireless Comm. Letters*, vol. 5, no. 6, pp. 564-567, Dec. 2016

[6] C. H. Bennett, G. Brassard, C. Crepeau and U. M. Maurer, “Generalized privacy amplification," in *IEEE Trans. on Info. Theory,* vol. 41, no. 6, pp. 1915-1923, Nov. 1995.