Computation-Communication Trade-off for Power and Bandwidth Savings in Remote Authentication over Wireless Networks

Arnab Paul Umakishore Ramachandran
College of Computing
Georgia Tech
Atlanta, GA 30332, USA

Abstract

We consider the problem of remote authentication over a long range wireless network using large signature keys such as biometric samples (fingerprint, retinal scans etc.). Because of the large size of these keys, and continual need for authentication, considerable power and bandwidth is consumed by such processes. We show that by using the concept of a holographic proof*, it is possible to significantly cut down the number of bytes transferred. While bandwidth savings is obvious, for long range wireless this means huge savings in power which is a critical resource for mobile devices. Our approach trades computaion for communication, the intuition being that power requirement for transmitting one byte over wireless is orders of magnitude higher than executing one instruction on a standard processor. Our simulations provide a detailed analysis of power savings under a reasonable energy-consumption model and indeed demonstrate the effectiveness of such approach.*

1 Introduction

As wireless communication is becoming *pervasive* along with computing, a new continuum of computation and communication is being born. Amongst a plethora of research issues engendered by this paradigm, security is quite critical. One can easily envision people accessing confidential documents from secured information stores onto devices such as palm-tops or cell-phones. Safe remote authentication is the key enabler of such ubiquitous accesses. The scenario differs considerably from the traditional networked computing systems: First, the devices in the new paradigm are mostly small and mobile; hence the connections are often reset and reestablished, resulting in fresh authentication activities. Second, as the new devices come more and more equipped with variety of sensors, it is natural to expect that confidentiality checks be performed at regular intervals without intervening the user activities; for example, while a person tries to access some financial documents in a busy airport through her cell-phone, the attached camera would be continually taking the picture of her face and would authenticate with the face recognition unit at the remote end ensuring that whenever the particular person stops using the phone, the right to access the documents is immediately revoked. Third, since the bandwidth is limited, such authentication protocols, however secure they might make the system, have to guarantee that they do not add substantial transport overhead. Which leads to probably the scarcest of the resources for mobile devices, viz., power. Since, long range wireless transmissions eat up batteries very fast, the challenge is to guarantee that a new security architecture such as this one, is not rendered useless by its power overhead.

In a foreseeable future, the primary mode of authentication will be sensor-based. One major advantage of such schemes over traditional password-based ones, is naturally the reduction in user intervention. However, another

important and often overlooked benefit is the size of the signature keys. While a password must be small for memorizing, sensor generated keys, such as biometrics are often quite large in size. High entropy offered by such keys makes it difficult for any adversary to reproduce them unlike password protection that is amenable to many attacks. In the light of the fact that authentication is done in continual fashion, the large key-size becomes even more critical from the viewpoint of power-consumption.

A potential solution to key length could be the use of cryptographic hash; instead of sending the whole key, one might consider sending a digest to be compared with the digest of the key stored at the remote end. But such a scheme fails in our case. The scenario we are considering here is subject to two factors that prevents the use of digests. First, sensors have inherent inaccuracy and sampling different. Therefore, two samples of the same object, such as a fingerprint, generated by two different sensors are most likely identical, although they are patterns close to each other according to some distance metric. Second, the conditions in which two samples are taken vary considerably over time; for example fingerprints of a person taken by the same device during different working conditions will most likely turn out to be different. One should therefore look for a *close match*, instead of identity. Cryptographic hash functions however, do not tend to preserve such metrics, two bit-strings slightly different could produce very different hash digests.

In this paper we present a technique that enables remote authentication over long range wireless in a continual fashion; even with keys of large size, this scheme results in significantly reduced consumption in bandwidth and hence power. The key ingredient in our approach is a concept of a *holographic proof*. Instead of providing a mathematical definition, we provide an informal description that would suffice for the purpose of reading this paper. A hologram is the imprint of an object (usually in 3 dimensions) such that every part of it contains a copy of the whole imprint; if a hologram is cut into two haves, each half will provide the full imprint of that object. Intuitively a holographic proof attempts to mimic this property. A proof (in our case, an authentication token) can be encoded so that almost every part of it carries the signature of this proof. Then, a verifier (in this case, a server authenticating the received token against a stored template token) can examine only a small segment of the proof and still be able to verify that the proof is correct.

How does one construct such a proof? Error correction codes provide a candidate infrastructure for constructing such proofs. Normally an error correction code expands a bit-string by adding *redundancy* so that even in the presence of corrupted, or missing bits, the original data can be recovered. Now, the primary task of authenticating is the following: One party presents an authentication token (s) to a server which matches this token against a stored template (s') to see if $s \approx s'$, i.e., if there is a close match between the two. However, one could reverse the question. Instead of asking if they are same (or close to each other), one can ask if the two strings are different. Now, if these strings are suitably encoded by adding redundancy, then it may be indeed possible to check that they are different by systematically examining only small parts of them. The success of this technique lies in the ability to achieve an appropriate encoding and subsequently to be able to examine the encoded versions. In this paper we present a simple encoding scheme that lends itself to a simple verification procedure. As a result, we also demonstrate how low-power (low bandwidth) continual fashion remote authentication can be enabled in long-range wireless systems. Ours is a scheme that resorts to randomization techniques, hence provides only probabilistic guarantees. However, to this end, we also demonstrate that these probabilities may be boosted sufficiently high (e.g., as high as $1 - 10^{-8}$) with only a few repetitions, incurring negligible effect on the power-savings.

As we have already mentioned, sensor-based samples can be quite diversified. They range from fingerprints, retinal scans, to voice recognition. It does not necessarily have to be a biometric sample either. The common verification procedure for all of them is some sort of pattern matching. The exact detail of the matching algorithm depends on the specific instance of the sample being considered. Unfortunately, it is not obvious how to construct an encoding scheme given an arbitrary pattern recognition problem. In this paper, we therefore focus

on a generic problem, that of computing the Hamming distance. The hamming distance between two strings is simply the number of bit positions they differ. It is one of the simplest distance metrics that lie at the heart of many pattern classification algorithms. For example, in a retinal scan match, retinal features are extracted in a binary feature vector and then two binary feature vectors are compared for their Hamming distance. For us therefore, the problem is abstracted into the following. A remote mobile device is presenting a token(s_U) on behalf of user U to a remote server. The server has in its database a template string (s'_U) as a sample token for U. Only if the hamming distance between the token and the sample is less than a specified threshold, access is granted.

The main motivation behind our scheme is the key observation : over long range wireless, transmission of one bit is equivalent to millions of processor cycles in terms of power consumption [6]. Conventional wisdom in this domain therefore has been to compute more and communicate less. Very recently, however this wisdom has been challenged by some researchers, mostly arguing in the line that architectural features in modern processors hinder the power savings expected from the reduction in communication [3, 1]. For example, in [1], the authors show that compressing files before releasing it onto the wireless, may end up increasing the energy needs, due to various factors such as the DRAM to cache ratio, cache miss rates *etc*. In [3], the authors argue that the energy consumed in refreshing the dynamic RAM associated with standard processors designed for mobile usages, such as Intel Strong-Arm, usually exceeds the wireless transmission energy and thereby making it energy-profitable to ship a block of data from a remote store instead of holding it in the DRAM for a long period of time. Our scenario however requires that the authentication token be in the main memory of a communicating device. Under a reasonable power model, we show that following our scheme results in saving power up to 70% of what would be required if the whole key is naively transmitted. A similar gain in bandwidth is obvious since the crux of the scheme is to reduce the communication.

One point to note here is that we have adopted a randomized scheme, and the guarantee of success therefore is probabilistic. However, this is not a major concern, since it is understandable that any deterministic scheme would require complete transmission of the keys and thereby not meeting the requirement of reducing bandwidth and power. A more important concern however, is to quantify and assure how much breach in security is being allowed as opposed to the power savings gained by such a scheme. In section 3 we will show that probability of admitting invalid keys can be made be substantially small. However, in a result pertaining to communication complexity, Pang and Gamal showed that the minimum number of bits needed to estimate the hamming distance correctly is $\Omega(n)$ [5]. Therefore, it is theoretically impossible to obtain a two sided ϵ-bound, *i.e.*,it is impossible to achieve an estimation $\hat{h}$ (the estimated hamming distance) of h (the original distance) so that $(1-\epsilon)h \leq \hat{h} \leq (1+\epsilon)h$ holds for all $0 \leq \epsilon \leq 1$ with arbitrary high probability. We have adapted the techniques presented in [2] that tries to safeguard against the underestimation. This is very important from the point of view of security. As a result of biasing the strategy towards overestimating the distance, some of the legitimate keys may be declared as false. Such cases are called false negatives. However, we show that although we cannot arbitrarily reduce the probability of having false negatives (p_{neg}, it is possible to keep it under an acceptable constant value. For example, having $p_{neg} \leq 0.2$, enables as many as 80% of the legitimated keys to be accepted, thereby applying the savings of bandwidth and power for 80% of the legitimate strings. This still boosts the overall power savings significantly. For the rest twenty percent, that fall victim as false negatives, once denied with the hamming signature, the whole key can be transmitted. In section 3 we discuss this issue quite elaborately.

The contributions from this paper are the following. Perhaps the most interesting observation we make is that certain randomized schemes could be used in the domain of mobile and ubiquitous computing as a means to reduce bandwidth and power. Next, although we have adapted a technique from [2], it is not possible to clearly determine theoretically the value of the each parameter that need to be set. To this end, we demonstrate an experimental methodology to bring the theory into practice. Moreover, the analysis in [2], had a limitation in that it tacitly assumed the hash function to be parity (although the technique inherently is open to any hash function). We

show that hash collision probability could be an important experimental knob that can be utilized set the systems parameters. We present a new analysis for deriving the bounds as a result of allowing arbitrary hash-collision probabilities. Finally, we quantify the bandwidth and power gains that can be achieved by such a scheme under a reasonable power model.

The rest of the paper is structured as follows. Section 2 describes the scheme in detail. In section 3 we carry out a complete evaluation. Section 4 derives the bound for estimating the hamming distance allowing arbitrary collision probability.

2 Protocol Description

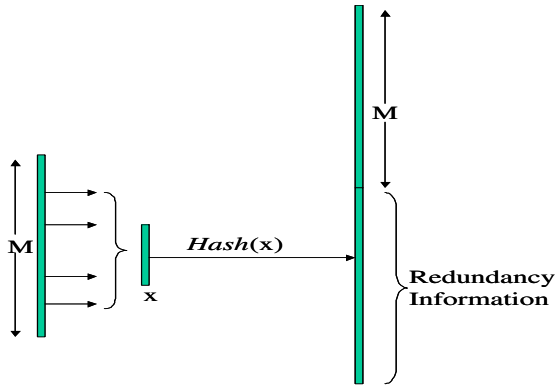


Figure 1. Encoding a message string M

In this section we describe the crux of the design of our protocol, i.e., the encoding scheme at the user end and the verification process at the server end. we start by describing the encoding.

2.1 Encoding at the user side

Figure 1 captures the encoding. M is the authentication token of length n that the user wants to communicate to the server. The encoding can be thought of as a standard coding scheme such as appending extra parity bits to the end of a data block. In the figure, x denotes a subset of bits taken from M (the bits in x are not necessarily consecutive in the original string M). The redundancy bits in the encoded string consists of hash values taken from all such strings x of all possible lengths. A simple example of hashing is taking the parity of x.

Figure 2 shows how a small subset is selected from the encoded string. The elements of R are the hash values of subsets x of M. Assume that they are ordered alphabetically and according to length of x. A small hash vector is constructed from R. A set of x-values are picked, say $x_1, x_2 \ldots x_t$, in the increasing order of their lengths, i.e., $|x_1| < |x_2| < \ldots < |x_t|$. And then from R, their hash values are collected into a small hash vector $[H(x_1), H(x_2) \ldots H(x_t)]$. This hash vector is sent over the wireless network instead of the original string M. The substrings are $x_1, x_2 \ldots$ are not chosen arbitrarily with the one restriction on their length. Their length increases exponentially, i.e., for some $\beta > 1$, (β is determined experimentally as we'll describe later) we choose their lengths as follows.

$$|x_1| = \beta \quad x_2 = \beta^2 \quad |x_3| = \beta^3 \quad \ldots \ |x_t| = \beta^t$$

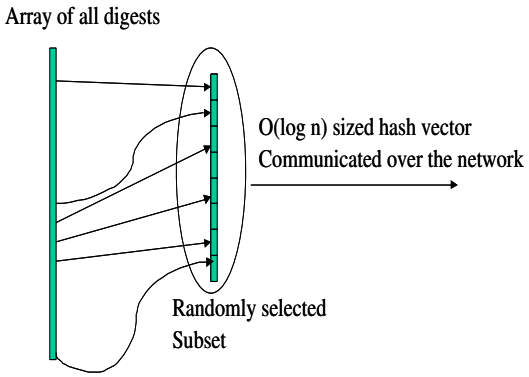


Figure 2. Creating the Hash vector

The longest such length that can be chosen this way is of course n, the length of M. Hence $t = \mathcal{O}(\log n)$, *i.e.*,the user sends a string of length exponentially smaller than the original string. For example, if n is 1 MB (2^{20}), and we use a hash of constant size, say 5 bytes, and $\beta = 2$, then the hash vector would be only 100 bytes long. Also note, that it is really not needed to compute the entire string R which is exponentially long. Instead, given a suitable random number generator, the user can randomly decide the substrings x_1, x_2 *etc*and then take their hash values to crate the hash vector. We assume that both the user and the server have access to the same pseudo-random generator. This is not a bottleneck as far as randomness is concerned. The two parties could exchange a random seed at the start-up of a transaction so as to randomize their choices properly.

2.2 Verification at the Server side

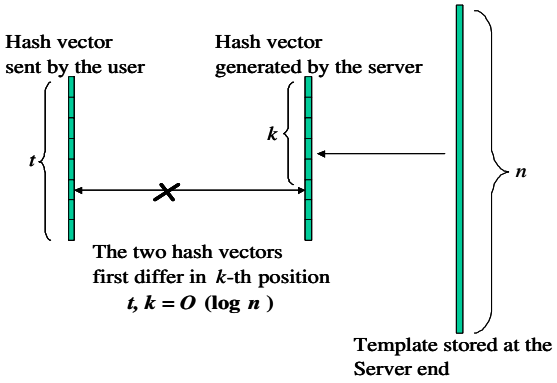
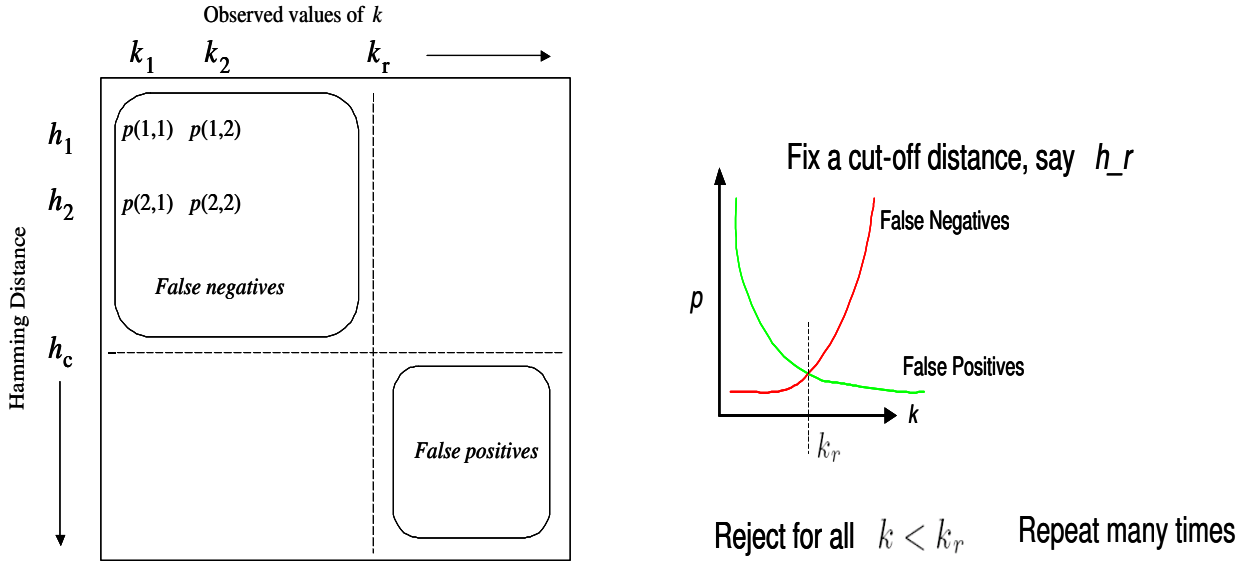


Figure 3. Comparison of Hash vectors

Figure 3 describes the actions taken on the server side. As already mentioned, we assume that the server and the user share the same pseudo-random generator. At the start up they exchange a random seed and initialize the generators in the same way. In the following steps they would generate exactly the same sequence of random numbers. Hence, the server has full knowledge of what substrings $x_1, x_2 \ldots$ *etc*were chosen by the user. The server also chooses exactly the same sequence of substrings and creates a hash vector in exactly the same way the user created her own hash vector.



(a) Probability Matrix for $k - h$ correlation

(b) False positives and Negatives

Figure 4. Visualizing False positives and Negatives in the correlation matrix and as function of k_r

At this point, the server compares the hash vectors at every coordinate. Notice that with increasing coordinates, a hash vector contains hash values for substrings of increasing length. If the original two strings, *i.e.,*the user token M and the server template T differ in many positions, there is higher chance that they would differ in a smaller substring picked from the same set of locations and therefore most likely would differ in their hash values as well. Intuitively therefore, for strings having large hamming distance, the value of k, which is an indicator for the first length of substrings that differ in hash values, would be small. And a high value of k would indicate that the strings are perhaps not too different from each other. Thus there is a strong correlation between k and the hamming distance of the strings being matched. In section 4, we establish this correlation theoretically. However, at this point we describe the experimental route that leads us towards making the decision on behalf of the server - *i.e.,*how to authenticate a user from the observed value of the parameter k.

2.3 Experimental determination of cut-off k

Once the parameter k is evaluated, the server takes its authentication decision based on this value. As we have already discussed in section 1, the server authenticates a string that is decided to be a close match to the template, *i.e.,*a string having hamming distance not more than a threshold set *a priori*. We call this threshold h_c. The decision process of the server is simple. It somehow chooses a cut-off k value, say k_r, so that for all $k < k_r$, the inference is that the corresponding hamming distance $h > h_c$. This is intuitive, as we have seen that the hamming distance and k values are correlated in an inverse fashion. Similarly, if $k \geq k_r$, then the server declares that the string in question has a hamming distance $h \leq h_c$ from the template and hence access is granted. Naturally, the choice of k_r decides the correctness and sensitivity of the decision. A choice of high k_r would be too conservative; while it will cut off almost all strings with $h > h_c$, at the same time it will not allow many legitimate strings that have $h \leq h_c$, yet, resulted in a lower value for k. We now describe how to make a choice of k_r on an experimental basis. Consider the matrix shown in figure 4(a). The rows are marked by increasing hamming distances. And

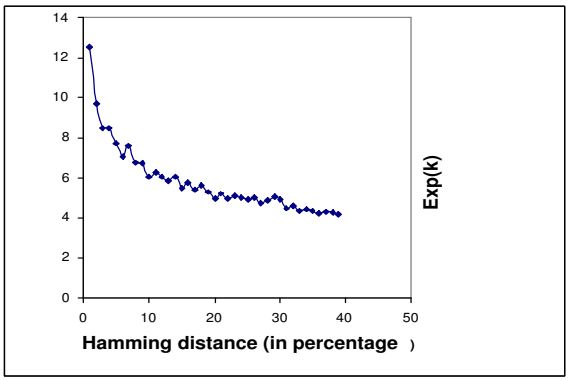
the columns are marked by increasing k values. The (i, j)-th entry in this matrix denotes the probability that a string of hamming distance h_i resulted in the observed value $k = k_j$. Observe that for a given row h_c and column k_r, the bottom right rectangle of the matrix therefore embody the total probability of *false positives*, *i.e.*,strings which in spite of having a larger hamming distance (than h_c) give rise to a larger k value (than the cut-off k_r). Similarly, the upper left rectangle of the matrix signifies the total probability for *false negatives*, *i.e.*,the strings having a low hamming distance yet rejected because of lower k values observed. Before deploying the system, it is reasonable to experimentally estimate these probabilities. Once this matrix is estimated, the choice is determined by the need of the application. For every combination of (h_c, k_r), there is a specific ratio of false positive and false negatives; the application can choose the cut-off for which this ratio suits the most. We will elaborate further on this in evaluation section; we will establish that by choosing so, the application really does not make a great deal of compromise in its security.

3 Evaluation

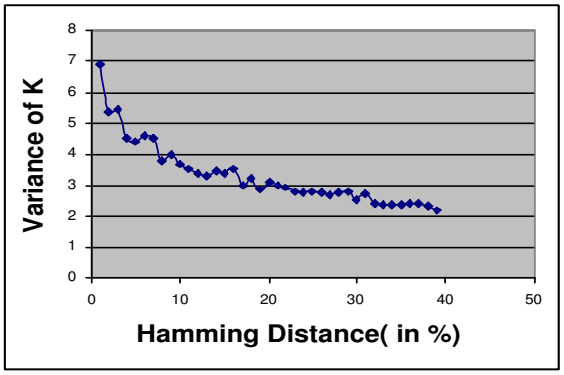
3.1 Security Assertions

In this section we present the evaluation of our methodology. As discussed in section 2, we have determined the systems parameters experimentally. The first step towards this is the evaluation of the correlation matrix (M) presented in figure 4(a). To do so, we have randomly generated many templates. For each such template, we generated a large number of proof strings of different hamming distances. We varied these hamming differences starting from 1% up to 40% of the length of the original string. For each proof string the hash vector is computed and the same is compared with the hash vector generated from the corresponding template string. This comparison results into an observed value of k. If the hamming distance is h, the (h, k) entry is incremented by one. Finally all entries are normalized between 0 and 1. Normalization, however, is optional. After M is determined, we determine the cut-off k. Note that this depends on the cut-off hamming distance. Typically, the applications generate legitimate proof strings that always lie within a certain distance from the original. Let's denote this distance percentage by h_1. This means, any string within percentage distance h_1, should be accepted with high probability. Similarly, there is another boundary, $h_2 \geq h_1$ such that any string with percentage distance more than h_2 should be rejected with very high probability. While the gap between the two values is application specific, a non-zero gap is quite natural. In other words, if a fingerprint that has a percentage difference of 5% with the template should be accepted with a high probability, another string with a distance of 6% should also be accepted with equal or slightly less probability. In summary, there should be a gradual fall in acceptance rate between h_1 and h_2; after h_2, the acceptance rate should become as close to zero as possible. We choose a cut-off hamming distance h_c, which is in the middle, $h_1 < h_c < h_2$. h_c could be chosen in many ways, as linear or non-linear function of h_1 and h_2. However, for our experiments, we have simply chosen as $h_c = (h_1 - h_2)/2$. Once, the cut-off hamming distance is fixed, the next task is to determine the cut-off k from the correlation matrix. We have already described this procedure in section 2.3.

Now we describe the main results of our experiments and demonstrate the validity of our methodology. We have performed our experiments with different string lengths and over varied range of other parameters. Figure 5(a) depicts the expected value of observed k as a function of the hamming distance in a typical experiment. This is generated from the normalized entries of the correlation-matrix. As expected, the expected value of k decreases with increasing percentage distance. This is due to the fact that strings with larger distance are quickly distinguished by samples of smaller sizes. Figure 5(b) presents the variance of observed k. Interestingly enough, the variance too decreases monotonically with increasing percentage difference. This means, for legitimate strings that are close match to the original template string, observed values of k are distributed over a larger range. This introduces the chances of having more false negatives. However, as the hamming distance increases, the variance



(a) Variation of $Exp(k)$ with h

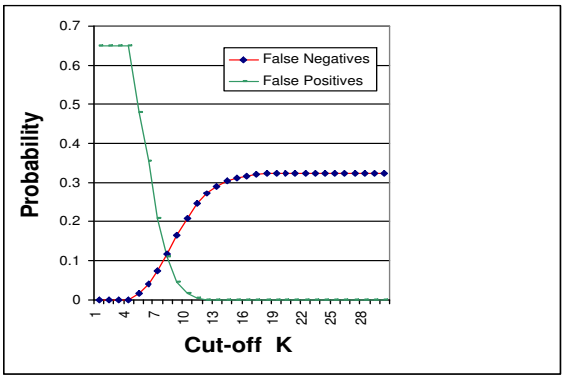


(b) Variation of $\sigma(k)$ with h

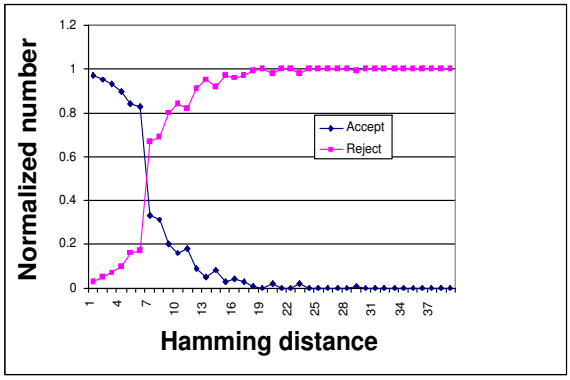
Figure 5. Variation of Expectation and the variance of observed value of k with the Hamming distance. For these experiments the string length was chosen as $n = 4096$ and $\beta = 1.3$

diminishes. The low variance at this end of the spectrum allows us to estimate more correctly whether a string is illegitimate enabling sharpening of the bound for false positives which is more important from security point of view.

We have experimented with several values of h_1 and h_2. We present a typical set of results just as a guideline of how things turn out for the experiments. Since this is a continuation of the same experiments that we described before, the string length is fixed at $n = 4096$ bits and $\beta = 1.3$. We set $h_1 = 6\%$ and $h_2 = 14\%$ and thereby $h_c = 10\%$. This choice is somewhat arbitrary. However, the overall pattern of our experiments is quite independent of the specific choice of h_c which is normally a requirement of the application. Figure 6(a) presents the false positives and negatives as a function of the cut-off k for $h_c = 10\%$. This means, if we set $k_r = 8$, we will have more than 20% false positives (unacceptable) and similar number of false negatives. Following our method, we set $k_r = 9$, that jointly minimizes the false positives and negatives. And then we repeat the experiments for $c = 5$ times. Repeating the experiment means preparing 5 different hash vectors on independent random trials. The string is accepted if $c_{majority} = 3$ of the hash vectors lead to the acceptance, *i.e.*,an observed value k value of greater than or equal to 9. The final message string will contain the collection of all five hash vectors. Figure 6(b) depicts the results after these repeated trials; the X-axis denotes the hamming distances and the Y-axis denotes the normalized number (of accepted and rejected strings respectively). It is observed that all strings with percentage distance less than 5, are accepted with high probability. Similarly, almost all the strings with distance more than fourteen percent, are rejected. How many bits are communicated finally? For these experiments, we used a hash with 16 buckets, *i.e.*,to represent a hash value we need 4 bits. We chose to sample up to a length β^{20}, i.e., the each hash vectors have 20 entries in it. Thus each hash vector is 80 bits long. And after five repeated trials the final string will be 400 bits. This is the final size to be communicated over the network. However, notice that as the string size increases, the hash vector length, being only logarithmic in the original string length, the communication ratio improves further.



(a) False positives and negatives as a function of cut-off K (k_r)



(b) Accepts and Rejects after five repeated trials

Figure 6. Determining k_c and results after repeated trials

3.2 Analysis of Power consumption

In this section we provide an analysis of the amount of power savings that can be achieved by deploying the proposed scheme. In general, such estimation can be quite complicated, the consumption of power being a function of many factors such as processor architecture (including the CPU and memory subsystems), the network link that involves analog components and also software factors such as the operating system, exact implementation of systems libraries and so on. Instead of stressing on fine-grain precision, we evaluate our figure of merit in a rather coarse-grain accuracy. The main power-drainers involved in this scenario are the processor, memory, and the network link. We compare two alternative methods; sending the whole data block naively over the wireless, and sending the hamming-distance sensitive hash-vector instead of the data block. Since, some computation is involved in the second case, we have to estimate the power consumed by the processor. The processing power consists of two parts : the energy consumed at the CPU by the instructions and the energy spent due to the cache-miss and memory accesses. Since the additional memory requirement of our scheme (over the baseline case) is very small, essentially just the size of the hash vector that would easily fit into the local cache of any modern processor, we can assume that the additional power consumption related to memory/cache can be kept out of the consideration and that we can solely focus on *instruction level power analysis*[7]. According to this scheme, energy consumed by each instruction is measured in isolation and then the total power is evaluated from the profile of the machine code. The consumption naturally varies from one instruction to another, however, one can consider an average case or worst case picture.

n	original size of the key
e_{proc}	average energy for executing one instruction
e_{net}	average energy to transmit one bit
N_i	number of instructions needed to compute one hash vector
b	bits needed to represent one digest
f	size of the hash vector
$v = fb$	size of one hash vector
τ	number of repeated trials
$I = \tau.N_i$	total number of instructions

Table 1. Parameters relevant for power-analysis

Table 1 denote the parameters of interest for computing the power consumption. The total power consumption for the first case (sending the whole key) is given by $E_1 = ne_{net}$. [1] In the second case, there are two factors, processing and transmission and the total energy is given by $E_2 = ve_{net} + Ie_{proc}$. Hence, the ratio of energy consumption in two cases, is given by

$$\frac{E_2}{E_1} = \frac{v\tau}{n} + \left(\frac{I}{n}\right)\left(\frac{e_{proc}}{e_{net}}\right)$$

Notice that $v = \mathcal{O}(\log n)$, and the fraction $v\tau/n$ falls sharply as the size n of the data block increases. For example, for a data size of 4Kb, hash bucket size $b = 4$ bits, $\tau = 5$ repeated trials, and a hash vector length $f = 20$, the length of the transmitted hamming sketch is $v\tau = fb\tau = 400$. According to an estimation provided in [6], transmitting one bit on the wireless over a distance of about hundred meters is equivalent to 3000 instructions

[1] Although there is some minimal processing required, such as byte copying, we assume the power is negligible. If anything such assumption favors underestimation of the savings that we predict

on a standard processor. Which means $e_{proc}/e_{net} = 1/3000$. Although this is a rough estimate and also couple of year old, the scenario has changed only in favor of low power processing. While long range transmission is limited by the physical factors beyond control, modern processors are becoming increasingly power efficient so far computing is concerned. At any rate, assuming $e_{proc}/e_{net} = 1/3000$, and noticing that it takes about 30000 instructions [2] to compute a Hamming sketch, we get

$$\frac{E_2}{E_1} = 0.1 + \left(\frac{30000}{4000}\right)\left(\frac{1}{3000}\right) = 0.10025$$

As n grows, the factor E_2/E_1 keeps constantly decreasing. As we have already seen in section 3.1, the total bandwidth and power savings actually depend on the false negatives; every time a legitimate string is rejected as false, the whole key is communicated in order to convince the verifier at the remote end. The fraction of such false rejections, as depicted by Figure 6(b) is less than 20%. Hence, amortized over many trials, the total energy ratio is given by

$$\frac{0.8E_2 + 0.2E_1}{E_1} = 0.2 + (0.8)(0.10025) \approx 0.28$$

Therefore, amortized over many transmissions, such a scheme results in bandwidth and power savings up to 72%.

4 Deriving the bound for arbitrary hash buckets

In this section we provide a theoretical analysis of what happens if the hashing is not just parity, and is allowed to take on any arbitrary hash bucket size. Table 2 lists the variables necessary for this analysis.

h	hamming distance
μ	hash collision probability
X	the proof sting
Y	the template string
X_l	l-th sample chosen from the proof string
Y_l	l-th sample chosen from the template string

Table 2.

We estimate the bounds on the estimation of h. Note that for any independent sample l, (of length β^l), the probability that the b-th bits of the template-sample and proof-sample is h/n. This is simply because the two strings differ in h/n fraction of bits. Hence we have the following set of equations :

Pr[$(\mathrm{X}_l)_b \neq (Y_l)_b$] $= h/n \quad \forall 1 \leq b \leq \beta^l$

Pr[$(\mathrm{X}_l)_b = (Y_l)_b$] $= 1 - h/n$

Setting $\delta = (1 - h/n)$, we get $Pr[X_l \neq Y_l] = 1 - (1 - h/n)^{\beta^l} = 1 - \delta^{(\beta^l)}$.

Now we estimate the error probability. μ is the probability of failure in a collision free hashing $H(x)$. To do so, we first observe the probability of having a collision in hashing for two distinct samples of length β^l. This the conditional probability given by $Pr[H(X_l) = H(Y_l)|X_l \neq Y_l]$. Now, observe from Bayes' Theorem

[2] by examining the machine code on an x-86 processor

$$\text{Pr [H(X}_l) = H(Y_l) | X_l \neq Y_l] = \frac{Pr[H(X_l)=H(Y_l) \cap X_l \neq Y_l]}{Pr[X_l \neq Y_l]}$$

The denominator on the right-hand side is the probability of false prediction and plugging in the values amounts to $\mu(1-\delta^{(\beta^l)})$. Hence the probability of not making a false prediction is $1-\mu(1-\delta^{(\beta^l)})$. Our objective is to tightly estimate h. However, since a two-way sharp bound is not possible, we focus on minimizing the false positives. We set the probability of no false prediction over k samples of geometrically increasing length very close to 1. That will allow us conservatively estimate the hamming distance.

$$\prod_l 1 - \mu(1 - \delta^{(\beta^l)}) \geq 1 - \epsilon$$

Approximating $1 - \mu$ with 1, we get

$$\prod_l [1 - \mu\delta^{(\beta^l)}] \geq 1 - \epsilon$$

At this stage, we need a first order Taylor series approximation. Set $\lambda = h/n$ and then $\delta^{(\beta^l)}$ becomes

$$(1 - h/n)^{(\beta^l)} \approx 1 - \lambda\beta^l$$

So the inequality becomes

$$\prod_l [1 - \mu\lambda\beta^l] \geq 1 - \epsilon$$

Set $\phi = \mu\lambda$. We get,

$$(1 - \phi\beta)(1 - \phi\beta^2) \ldots (1 - \phi\beta^k) \geq 1 - \epsilon$$

Further approximating with the product terms, we set,

$$(1 - \phi\beta)^k \geq 1 - \epsilon$$

Or in other words, we get a bound on k, the first sample length we see any difference of the hash values as a function of the general discrepancies of the two strings. taking Logarithms on both side we get,

$$k \geq \frac{\ln(1 - \epsilon)}{\ln(1 - \phi\beta)}$$

And substituting the values for different quantities and further simplifying we get a bound on h, the hamming distance as a function of k. This gives us an way to estimate h.

$$\frac{h}{n} \leq \frac{1 - (1 - \epsilon)^{1/k}}{\mu\beta}$$

The above equation tells us that our method underestimates h with no more than a failure probability ϵ. This ϵ is a free parameter and can be set to any user-specified value. Depending on the required success probability, we have to set the length of the hash vector. The bounds can be further improved with repeated trials; this can be shown by a Chernoff-bound type argument. For such a tehnique the reader can consult [4].

References

[1] K. Barr and K. Asanovic. Energy aware lossless data compression. In *1st International Conference on Mobile Systems (MOBISYS)*, 2001.

[2] G. Cormode, M. Paterson, S. C. Sahinalp, and U. Vishkin. Communication complexity of document exchange. In *Symposium on Discrete Algorithms*, pages 197–206, 2000.

[3] Fryman, C. Huneycutt, H. S. Lee, K. Mackenzie, and D. Schimmel. Energy efficient network memory for ubiquitous devices. *IEEE MICRO*, XX(YY), September-October 2003.

[4] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[5] K. Pang and A. El-Gama. Communication complexity of computing the hamming distance. *SIAM Journal on Computing*, 15(4):932–947, 1986.

[6] G. J. Pottie and W. J. Kaiser. Embedding the internet: wireless integrated network sensors. *Communications of the ACM*, 43(5):51–58, May 2000.

[7] V. Tiwari, S. Malik, A. Wolfe, and M. Lee. Instruction level power analysis and optimization of software. *Journal of VLSI Signal Processing, Kluwer Aacd. Publisher, Boston*, pages 1–18, 1996.