CONCEALED KEY-PHRASE VERIFICATION

Juan M. Huerta

IBM T. J. Watson Research Center
Yorktown Heights, NY

ABSTRACT

It is a common task in conversational applications to validate a claimed identity through utterance verification mechanisms; for example, by asking for a password or a challenge question. Sometimes, due to the private nature of the keyphrase, it is desirable to impose restrictions on how this keyphrase, or information that leads to it, is handled during the verification process. In this paper we describe a method where the utterance verification takes place in a *concealed* way, i.e., in a way that the keyphrase cannot be inferred or obtained directly by an intruder, from the decoding and application artifacts. We achieve this by a novel algorithm based on matrix operations in the verification network's nodes. If the traversed path corresponds to the keyphrase, the annotation converges to the dominant eigenvector of the occluded kernel matrix.

1. UNPROTECTED KEYPHRASE VERIFICATION

User verification is an often performed task in speech applications. When the application takes place in a distributed or network environment (e.g., in a telephony environment [1]) the application designer needs to take into consideration the possibility that the confidentiality of the data in the network or in the application servers is vulnerable to network eavesdroppers, intruders, or even malicious application analysts (we assume that the intruder is in the network and is not an eavesdropper in the telephony circuit). If the network traffic is not secure or encrypted, the intruder will be able to observe the sequence of request and response cycles between the application components (e.g., the voice browser, in scenario [1]) and analyze markup, submitted values, and application artifacts; even if the communication is encrypted, a typical speech application leaves behind a substantial set of resources and artifacts (cached grammars, lexica, log files, etc.) which can be collected and reverse engineered to extract or infer the authentication information.

Due to industry trends like business process outsourcing and application hosting, and to the emergence of sophisticated analysis tools, more and more information is made available by the application for system analysts and tools to perform their jobs. The information that makes the application more analyzable but it also makes it more prone to be reverse engineered (e.g. [2]).

A common way to perform user verification, without using biometrics, is to verify a password or a multi-word utterance (the *keyphrase*, denoted by u_k) for a specific user $user_k$ under the assumption that only this user will know it. Examples of what can constitute a good keyphrase are: date of birth, social security number, amount of last account transaction, etc. This process is called text-dependent user verification.

A basic keyphrase verification algorithm can be formulated as follows: let $\hat{u}_k$ be the recognition hypothesis to the challenge question specific to $user_k$, and $\hat{a}_k$ be its semantic annotation using the annotation grammar A (i.e., $\hat{a}_k = A(\hat{u}_k)$). This process can be done concurrently by the decoding verification grammar G, which is the combination [3] of a decoding network D and the semantic annotation network A. Keyphrase validation is performed in terms of a distance computation between $\hat{a}_k$ and a_k. This idea can be further elaborated for enhanced efficiency, robustness and flexibility [4].

In an unprotected system, an intruder can obtain for a specific session $\hat{a}_k$ and A, and based on session information can associate this information with $user_k$, and can compute A^{-1} and infer $\hat{u}_k = A^{-1}(\hat{a}_k)$. For example, the annotation "A-03-75" might be traceable to "august third 1975" if A is available and easily invertible. In an unprotected system, an intruder might be able to associate this date as the birthday of user "John Doe".

The above scenario provides a motivation for information concealment so that an intruder cannot easily compromise the verification process for a $user_k$ by observing any of the application information or network traffic. We present a method for concealing keyphrase verification; in our method, a verification grammar and a key are dynamically generated each time a keyphrase verification task is performed, and in which the semantic annotation takes place in the form of matrix-vector operations. If the keyphrase is correct, the annotation will converge to the key. In this way, the client can generate annotations robust to reverse engineering. An intruder capable of observing network traffic, voice browser markup, annotation grammars, annotation scripts, and the resulting annotation hypothesis will not be able to infer the keyphrase. In the next two sections we describe our method in detail.

2. CONCEALING VERIFICATION

2.1. Desirable Features

In order to prevent an intruder with capabilities like the ones described in section 1 compromise the keyphrase of $user_k$, it is desirable that a system has the following features:

- The correct keyphrase u_k or the path traversed by the correct hypothesis in the decoding graph G cannot be inferred statically by looking at the resulting annotation $\hat{a}_k$ or by statically analyzing the graph (A, G, or D).
- The correct keyphrase u_k cannot be reverse-engineered or obtained from decoding artifacts, including graphs, grammars, logs, annotations, session information, network traffic etc.
- The probability of finding the u_k by trial and error (even after observing the user authenticate in multiple instances) is low.
- The annotation function A provides an annotation $\hat{a}_k$ is a smooth function of u_k, i.e., small deviations from the correct utterance result in small distances between hypothesized and true annotations, and so forth.
- Having observed several times the keyphrase's annotation for $user_k$ in various instances would not be of help to produce a correct annotation for $user_k$ in session n.

2.2. A Structured run-time environment

To facilitate the above features, we propose structuring conversational applications in a way that decouples the security handling subsystem from the interaction subsystem [5]. Thus the application will have two parts: a secure and an unsecure part. We assume that the security handling subsystem (the secure part of the application) is responsible for the generation of the key and verification grammar G_k from the cleartext keyphrase for each user k; while the interaction part of the application (the unsecure part) is responsible for handling the user's response, obtaining its annotation and submitting this annotation to the secure part of the system, where the comparison between key and annotation is performed.

The secure part can be a commercial identity management solution, while the unsecure part corresponds to the voice browser, the recognition engine, and application components that handle interaction (server pages etc.).

The security subsystem can make G not only user dependent, but also session dependent, thus $G_{k,n}$ will be particular to every user k and every session n. They key and the annotation will also be session dependent. We will describe later on how to generate $G_{k,n}$. This grammar needs to be a one-way function, i.e., it will be hard to find $u_{k,n} = A^{-1}(u_{k,n})$ given $u_{k,n}$ and $G_{k,n}$.

As we said, the unsecure part of the application takes $G_{k,n}$ and performs recognition of the utterance of the grammar, and obtains and annotation $\hat{a}_{k,n}$ which is compared against the key $a_{k,n}$ in the secure part of the application. The keyphrase annotation $a_{k,n}$ (the key) is the annotation of utterance u_k when grammar $G_{k,n}$ is used, and because we assume $G_{k,n} \neq G_{k,m}$ when $n \neq m$, (i.e., the grammar changes in every session for every user), knowing the annotation $a_{k,n}$ for user k during session n is of no use in session m.

The question is how to generate G_n so that $\hat{a}_{k,n}$ has a smooth error function, and G_n is robuts to reverse engineering. We now discuss a method to produce these grammars.

3. CONCEALING KEY-PHRASE IN GRAMMARS

3.1. Verification Grammar Essentials

The task of keyphrase verification has been previously studied (e.g., [4]). We briefly describe here some previously established concepts that are important in our analysis.

Verification can be carried out as recognition, with a particular type of grammar: a verification grammar. We define a verification grammar G as a decoding graph in which the keyphrase u_k can be recognized, resulting in a semantic annotation a_k. In [4] the authors describe a hypothesis testing method for verification grammars, as a way to accept the user's utterance as a keyphrase match. A keyphrase verification grammar needs to contain *anti-subword models* [4] with their corresponding semantic interpretation.

In our paper we assume that the perplexity of the recognizer is high in order to prevent intruders to exhaustively try the list of entries in the grammar G.

3.2. Embedding vector operations in network nodes

We describe our approach to substituting simple semantic annotations in a verification network G for node vector operations on a given vector.

Let path $P_k = \{p_1, p_2, ..., p_L\}$ with $0 \leq j \leq L$, denote the path of nodes traversed in annotation network A by the observed utterance $\hat{u}_k$.

Let network A be of size Z(i.e., have Z nodes) and with each node $N_i \in A$ we associate a vector $Y_i \in \Re^{dxd}$.

Let seed vector x_0 be a random vector of dimension d.

The annotation performed by A corresponding to an observed utterance $\hat{u}_k$ will be $\hat{a}_k$. Assume that $\hat{u}_k$ results in traversed path P_k when A is used, then $\hat{a}_k$ is computed in the following way:

$\bar{x}_{k+1} = Y_k \bar{x_k}$ with$node_k \in P$ and $0 \leq k \leq L$;

and $\hat{a}_k = \bar{x_L}$;

In other words,

$\hat{a}_k = \prod(Y_i x_0)$;

Thus, A is an annotation grammar which instead of having words or symbols in the nodes, performs a matrix multiplication on a input vector x_0 and returns as annotation the vector $\hat{a}_k$.

Verification is thus performed by computing the distance between the annotation and key vectors:

$d = cosine\_distance(key, \hat{a}_k)$

3.3. Properties of the annotation vector when all nodes in the verification path share matrix K

Let matrix K be a symmetric matrix. Let the nodes in the keyphrase path P share K as their annotation matrix. Then $\hat{a}_k$, as defined in the previous section, will converge to the eigenvector corresponding to the largest eigenvalue of K (the dominant eigenvector). This is a fundamental principle in linear algebra (power method [6]).

As long as the dominant eigenvalue of K is unique and the inner product of the key vector and x_0 is not zero (a condition that the secure subsystem should verify when choosing K), then $x_L \to key$ as $L \to \infty$ linearly with convergence factor $max\{\lambda_{d-1}/\lambda_{d,}, |\lambda_1|/\lambda_d\}$.

We can choose K so that it results in a desirable rate of convergence, given L (the length of the path P). We will refer to matrix K as the *kernel of the keyphrase path* and its dominant eigenvector becomes the key of K. Additionally, this algorithm allow us to use any real vector as the seed vector x_0 in the annotation process and the annotation will converge to the key if P is traversed.

The seed will converge to the key with the rate described above. The convergence rate is a function of the distribution of the eigenvalues of the kernel. Given a random seed, and a rate of convergence we will achieve an arbitrary distance to the key with a probability inversely proportional to the L. Thus, we need to balance then the target chain length and the choice of kernel K as to obtain the desired probability of false acceptances and false rejections. For shorter chains (e.g., zip codes) we want the rate of convergence $\hat{a}_k$ to converge to the key K faster, for longer chains (e.g., social security numbers) we want it slower, as premature convergence will imply that we are only using prefixes of the keyphrase for verification. Manufacturing K to have real eivenvalues with desired convergence rates is a simple application of the Spectral Theorem. Additionally, K should be chosen as to ensure stability of the computation of its eigenvectors.

Obviously, the nodes that are not in the keyprhase path can be chosen in a way that their eigenvalues corresponding to their latest eigenvectors are not close, from a cosine-distance perspective, of the key of K. These matrices are responsible of *disrupting* the convergence process of x_i.

3.4. Random-Seed Verification Grammars

In section 2.1 we determined that an intruder that has observed several times the keyphrase's annotation (or key) for $user_k$ in various sessions should not be able to easily produce the correct annotation for $user_k$ in session n. This means that even for a specific user, K (the keyphrase kernel) needs to change in every session so that the key of K (and the guessed $\hat{a}_k$) changes at every time.

Given that K and x_0 are generated randomly every time we perform verification, we can expect that x_0 will allow for correct phrase verification with a given probability. When P is traversed correctly then $\hat{a}_k = K^L * x_0$. The probability of a correct Verification is then,

$P(cos\_distance(key, K^L * x_0) > Threshold)$.

It is matter of manipulating the eigenvalues of K to ensure that the probability defined above is acceptable given the security requirements of our application.

3.5. Concealing the kernel Matrix via occluding Matrices

If we provide A as a verification grammar, as we have so far discussed, it becomes trivial to an intruder to hypothesize the correct utterance easily: find the node matrices that repeat themselves in connected paths from beginning to end nodes most frequently and try their dominant eigenvalues as keys (A in our method appears L times). It is necessary to *occlude* K, which effectively means that all the node grammars have to look different to an observer, yet the correct path will make the seed vector converge to the key vector.

We achieve this by providing the following annotation matrices to the nodes of A as follows:

For nodes in the path P,

$Y_i = G_{i-1}^{-1} K G_i$ when $Y_i \in P$ and $2 \leq i \leq L-1$,

$Y_1 = K G_1$ when $Y_1 \in P$.

$Y_L = G_{L-1}^{-1} K$ when $Y_L \in P$,

And for nodes *not* in the path P,

$Y_j = G_j$.

Where all G_i and G_j are random, non-symmetric matrices. The occluding matrices (G_i, occurring in P) need to be invertible.

Effectively, what we have done is introduce matrices and their inverses that cancel each other (if the path is P) and leave just the kernel matrix in the path.

An intruder who is trying to reverse engineer the grammar would have to find the path in A of length L which is factorizable as follows:

$(XG_1)(G_1^{-1}XG_2)(G_2^{-1}XG_3)\cdots(G_{L-2}^{-1}XG_{L-1})(G_{L-1}^{-1}X)$

For such path, the intruder would postulate X as the kernel, and propose its dominant eigenvector as the key and save the path associated with $user_k$. Even if such factorization was at all feasible, which is not practical, the secure subsystem can generate a grammar with a large number of false paths which factorize as above. The correct path would only be one of many paths with this type of factorization.

3.6. Putting it all together

When an application needs to verify that the current user knows the keyphrase utterance for $user_k$ the following steps ensue:

- The secure subsystem retrieves the keyphrase and assembles a general verification network A.
- The secure subsystem generates symmetric random vector K with target eigenvalue distribution and obtains its dominant eigenvector, the key

- The secure subsystem associates matrices in each node in A: occluded K for nodes in P and random matrices otherwise.
- The secure subsystem publishes A and the challenge question to the unsecure subsystem, which in turn makes these information available to the interaction component.
- The user provides an utterance to the challenge question, which determines an annotation path A. The annotation algorithm generates a random seed and traversing the path performs the matrix operations of the path in the random seed.
- The resulting vector is sent to the secure subsystem which computes the cosine distance between key and received vector. It determines with a certain confidence whether or not the user uttered the keyphrase.

An intruder would need to find the inverse of A given the resulting vector, or statically perform the factorization described in section 3.5, both computationally impractical.

4. EVALUATION

In this section we present the results of a simulation experiment in which we evaluate different verification configurations assuming no recognition errors (recognition errors will mostly impact the false rejection rate which does not compromise security but affects user experience). We evaluate premature acceptance rates given various false rejection rates. We also observe the tradeoff that exists between computation time and probability of error.

Table 1 shows our simulation experiment results. Each row correspond to a specific condition. For each condition we generated 1000 verification trials. A verification trial corresponds to the generation of a random kernel matrix, adjusting its eigenvalues to match the target ratio, generate a random seed vector, and compute the output vector given the decoding path, and evaluate the cosine distance for each prefix of the path traversed between the annotation vector and the key; then, for each condition compute the distance threshold that results in a desired false reject rate of all the trials (1,2,5 and 10%) and using that threshold count how many times we reached this threshold prematurely (before Phrase Length). Premature acceptance rates are shown in columns 5 to 8. The conditions parameters in the table are as follows: column 1 corresponds to the size of the keyphrase (5, 7 and 9, e.g., zip codes, telephones and social security numbers), column 2 corresponds to the dimensionality of the matrix, column 3 corresponds to the ratio between the dominant and smallest eigenvalues, column 4 shows the time (in CPU units) to run 1000 scenarios, and columns 5 through 8 the premature acceptance rates for the 4 distance threshold levels that result in 1%, 2%, 5% and 10% false rejection rates respectively.

We observed that the dimensionality of the matrix has an impact on the computation time (as finding the eigendecomposition of large matrices is more expensive), however, for the same eigenvalue ratio the false acceptance rate of large matrices is smaller than for smaller matrices. The false acceptance rate is larger for long chains given false rejection rate. Overall the most secure choice is large matrices with large eigenvalue ratios and large verification chains, which result in very little premature false acceptance, even with very low false reject rates. This, of course, corresponds to the most computationally expensive cost.

Table 1. Simulation Results

Phr.	Mat.	Eig.	Time	PA w.			
Len.	Dim.	ratio		FR=1%	FR=2%	FR=5%	FR=10%
5	9	1.7	0.98	17.0	15.0	9.7	7.3
7	9	1.7	0.99	11.6	10.2	7.6	5.4
9	9	1.7	1.08	8.3	7.3	5.6	4.1
5	15	1.7	1.77	9.2	6.6	4.5	2.7
7	15	1.7	1.77	6.2	4.7	3.3	1.8
9	15	1.7	1.86	5.6	3.6	2.4	1.6
5	15	2.9	1.62	0.3	0.2	0.1	0.1
7	15	2.9	1.76	0.2	0.1	0.0	0.0
9	15	2.9	1.81	0.4	0.2	0.1	0.1

5. CONCLUSION

We have shown a method to conceal a path in a graph that performs vector operations in its nodes. We have described how this is a robust method for a verification system in which a secure subsystem in an application does not want to disclose a keyphrase to a non-secure subsystem in the application. The unsecure system will not be able to reverse engineer this graph to obtain the keyphrase or the key, even from correct annotations. This can be a solid foundation for keyphrase verification systems, particularly in environments in which privacy is a concern (e.g., hosting, outsourcing, SOA etc.).

6. REFERENCES

[1] B. Lucas, "Voicexml for web-based distributed conversational applications," *Communications of the ACM*, vol. 43.

[2] A. Marchetto C. Bellettini and A. Trentini, "Webuml: reverse engineering of web applications," in *Proceedings of the 2004 ACM symposium on Applied computing*, Nicosia, Cyprus.

[3] M. Mohri, F. Pereira, and M. Riley, "Weighted finite state transducers in speech recognition," in *ISCA ITRW Automatic Speech Recognition: Challenges for the Millenium.*

[4] T. Kawahara, C.-H. Lee, and B.-H. Juang, "Key-phrase detection and verification for flexible speech understanding," in *Proc. ICSLP '96*, Philadelphia, PA, vol. 2.

[5] B. Pfitzmann and M. Waidner, "Federated identity-management protocols," in *11th International Workshop on Security Protocols*. Springer-Verlag.

[6] B. N. Parlett, *The Symmetric Eigenvalue Problem*, Prentice-Hall Series in Computational Mathematics, Englewood Cliffs, NJ, 1980.