Environmental Key Generation towards Clueless Agents

James Riordan[*]
School of Mathematics
University of Minnesota
Minneapolis, MN 55455
riordan@math.umn.edu

Bruce Schneier
Counterpane Systems
101 E Minnehaha Parkway
Minneapolis, MN 55419, USA
schneier@counterpane.com

Abstract. In this paper, we introduce a collection of cryptographic key constructions built from environmental data that are resistant to adversarial analysis and deceit. We expound upon their properties and discuss some possible applications; the primary envisioned use of these constructions is in the creation of mobile agents whose analysis does not reveal their exact purpose.

1 Introduction

Traditional cryptographic systems rely upon knowledge of a secret key to decipher messages. One of the weaknesses induced by this reliance stems from the static nature of the secret keys: they do not depend upon temporal, spatial, or operational conditions. By contrast, the secrecy requirements of information are often strongly linked to these conditions. This disparity is, perhaps, most easily seen and most problematic in mobile agents.

Mobile agents, by nature, function in and move through a wide variety of environments. Security properties of these environments vary greatly which creates problems when the mobile agent needs to carry security sensitive, or otherwise private, material. If the agent passes through an insecure network it may be analyzed so that any information carried by the agent becomes available to the attacker.

This information is often private in nature, and cannot easily be disguised. For example, an agent written to conduct a patent search will reveal the nature of the information desired. This information may, in turn, reveal the intentions of the requester of that patent search.

The problem is somewhat similar to the one faced by designers of smart cards which keep secrets from the card's carrier. To solve that problem, smart card designers have developed a number of techniques to make the hardware either tamper-resistant or tamper-evident thereby protecting the secret. Unfortunately these techniques are not applicable to mobile agents due to the fact that software, unlike hardware, is completely and trivially observable.

[*] The first author is now with the IBM Zurich Research Laboratory in Switzerland

To address this class of problem we introduce the notion of *environmental key generation*: keying material that is constructed from certain classes of environmental data. Using these keys, agents could receive encrypted messages that they could only decrypt if some environmental conditions were true. Agents with data or executable code encrypted using such keys could remain unaware of their purpose until some environmental condition is met.[1]

Environmental key generation is similar to the idea of ephemeral keys: keys which are randomly created at the time of use and destroyed immediately afterward. Public-key systems for encrypting telephone conversations—the STU-III [Mye94], the AT&T TSD [ATT92], the Station-to-Station protocol [DOW92]—make use of this idea. The communication model is different, however. Ephemeral keys are used when two parties want to communicate securely at a specific time, even though there is no secure channel available and the parties have not previously negotiated a shared secret key. Environmental key generation can be used when the sender wishes to communicate with the receiver, such that the receiver could only receive the message if some environmental conditions are true. Environmental key generation can even be used in circumstances where the receiver is not aware of the specific environmental conditions that the sender wants his communication to depend on.

The difficulty with constructing an environmental key generation protocol is that the threat model assumes that an attacker has complete control over the environment. All information available to the program is available to the attacker, all inputs to the program are supplied by the attacker and the program state itself is completely determined by the attacker. As such, the constructions must resist direct analysis and dictionary attacks in the form of Cartesian deception (in which the attacker lies about the environment).

Ultimately, if the attacker has access to both the agent and the activation environment (the environment in which the agent can construct its keys) then he will have access to all secret information as well. This is often not a problem.

In this paper, we propose three basic approaches toward securely generating cryptographic keys from environmental observations. The first involves direct manipulation of the environment in a specific and cryptographically unspoofable manner. The second involves reliance upon a partially trusted server. The third uses the obfuscation of the nature or value of the environmental data being sought through the use of one-way functions.

2 Clueless Agents

In the basic construction, an agent has a cipher-text message (a data set, a series of instructions, etc.) and a method for searching through the environment for the data needed to generate the decryption key. When the proper environmental

[1] These agents might reasonable be likened to the sleeper agents of "The Manchurian Candidate" and other Cold War era spy films.

information is located, the key is generated, the cipher-text is decrypted, and the resulting plain-text it acted upon. Without the environmentally supplied input, the agent cannot decrypt its own message (i.e. it is *clueless*), and can be made cryptographically resistant to analysis aimed at determining the agent's function.

Let N be an integer corresponding to an environmental observation, $\mathcal{H}$ a one way function, M the hash $\mathcal{H}$ of the observation N needed for activation, $\oplus$ the bitwise *exclusive-or* operator, *comma* the catenation operator, R a nonce, & the bitwise *and* operator, and K a key. The value M is carried by the agent.

One way functions can be used to conduct tests and construct the keys in a way that examination of the agent does not reveal the required environmental information. A number of such constructions are possible:

- if $\mathcal{H}(N) = M$ then let $K := N$
- if $\mathcal{H}(\mathcal{H}(N)) = M$ then let $K := \mathcal{H}(N)$
- if $\mathcal{H}(N_i) = M_i$ then let $K := \mathcal{H}(N_1, ..., N_i)$
- if $\mathcal{H}(N) = M$ then let $K := \mathcal{H}(R_1, N) \oplus R_2$

The constructions differ in types of data which are most naturally provided as input or in the programmer's ability to determine the output (as needed by a threshold scheme 6.2); the important feature of each is knowledge of M does not provide knowledge of K.

This general sort of construction is not uncommon. The *if* clause of the first construction is used in most static encrypted password authentication schemes (e.g. Unix). As with static password schemes, dictionary attacks present a problem. The fact that an agent may pass through or even execute in a hostile environment compounds this problem greatly (as would publishing your password file). None the less, several useful and cryptographically viable constructions are possible.

3 Basic Constructions

The very simplest clueless agents look for their activation keys on a fixed data channel. Example channels include:

- Usenet news groups. A key could be embedded in any of several places in a (possibly anonymous) posting to a particular newsgroup. It could be the hash of a particular message, or the hash of a certain part of the message.
- Web pages. Likewise, a key could be explicitly or steganographically embedded in a web page or image.
- Mail messages. The message could contain a particular string that would serve as a key, or the key could be a hash of a message.
- File systems. A key could be located in a file, the hash of the file or the hash of a particular file name.

– Local network resources. A key could be generated as the hash of a local DNS block transfer or as a result of a broadcast ping packet. Threshold schemes would be particularly valuable with
schemes like this.

If, for example, the agent knows that its activation key will be posted to a particular newsgroup, it would continuously scan the newsgroup looking for a message N such that $\mathcal{H}(\mathcal{H}(N)) = M$. An attacker would know that N would be posted to the newsgroup, but would need to see the message N before he could construct the key, $(\mathcal{H}(N))$, and thus figure out the agent's purpose.

The nature of the data channel determines the utility and properties of the construction based upon that channel. This nature includes:

– Who can directly or indirectly observe the channel?
– Who can manipulate parts or the whole of the channel?
– Along what paths does the channel flow?
– How do observations of the channel vary with the observer?

This abstract notion is best explained by a few diverse examples.

3.1 Example: Blind Search

We take the data channel to be an online database containing a list of patents with an online mechanism for executing search agents. The channel does not have a particularly interesting nature but yet generates an interesting agent.

We suppose that Alice has the an idea that she would like to patent. She wishes to conduct a patent search yet does not wish to describe her idea to the owners of the database search engine. This desire can be realized through the use of a clueless agent.

To make matters concrete, we assume that Alice's idea is to build a smoke detector with a "snooze alarm" so that she can temporarily de-activate the alarm without unplugging it.[2]

She begins by computing:

1. $N :=$ a random nonce
2. $K := \mathcal{H}($"smoke detector with snooze alarm"$)$,
3. $M := E_K($"report findings to alice@weaseldyne.com"$)$, and
4. $O := \mathcal{H}(N \oplus$"smoke detector with snooze alarm"$)$.

She then writes an agent which scans through the database taking hashes of five word sequences

[2] This would be a very useful item for the kitchen; smoke alarm manufacturers take note.

– **for** five word sequence (x) **in** the database **do**
– **if** $\mathcal{H}(N \oplus (x)) = O$ then **execute**$= D_{\mathcal{H}(x)}(M)$

The agent can now search through the database for references to "smoke detector with snooze alarm" without actually carrying any information from which "smoke detector with snooze alarm" could be derived.

In this example, if the owner of the database is watching all search agents in an attempt to steal idea, he will only observe a description of Alice's idea if he *already* has a description of the idea. Methods of rendering this scheme less sensitive to different wordings are discussed in Section 6.

3.2 Example: Intrusion Detection

One of the most problematic aspects of intrusion detection is that wide scale deployment of a particular method tends to limit its effectiveness. If an attacker has detailed knowledge of the detections system installed at a particular site, he is better able to avoid its triggers. As such, it would be better to deploy an intrusion detection system whose triggers are not easily analyzable. Environmental key generation could be used to encrypt sections of the intrusion detection's executable code until such time as a particular attack is executed.

While in this case the attacker could easily stage a dictionary attack by mimicking the LAN's behavior, such a simulation would require extensive knowledge of the LAN. Acquisition of that knowledge would likely to trigger the detection system.

3.3 Example: Network-Based Operation

It is often desirable to have an agent which can only run in certain environments. The agent may be collecting auditing data on certain types of machines or certain points in the network. It may need to carry out an electronic commerce transaction, but only from within the network of a certain vendor.

An interesting, although malicious, application of this sort of construction would the creation of a *directed virus*. Such a virus could carry a special set of special instructions which could only be run in a certain environment. The novelty of this construction is that examination of the virus without explicit knowledge of the activation environment would not reveal its "special instructions".

We suppose that Alice wishes to write a virus which should carry out the instructions if it finds itself inside weaseldyne.com so that it would be infeasible to determine what the special instructions are without knowing it is keyed for weaseldyne.com. Alice finds the name of a machine on the inside of weaseldyne.com's network. She does so through some combination of examining mailing list archives, social engineering, and assumptions about naming conventions (e.g. there is often a *theme*).

We will assume that the name of the machine is pooky.weaseldyne.com. She computes:

1. $K = \mathcal{H}(\text{“pooky.weaseldyne.com''})$
2. $M = E_K(\text{“report findings to alice@competitor.com''})$

She then writes a virus which, when activated, requests local DNS information and applies $\mathcal{H}$ to each entry looking for its key.

In this example, staging a dictionary attack is already quite difficult. Methods of making it yet more difficult are discussed in section 6.

4 Time Constructions

The time-based constructions allow key generation based on the time. These constructions rely upon the presence of a minimally trusted third party to prevent a date based dictionary attack. The third party is minimally trusted in the sense that it does not need to know either of the two parties nor does it need to know the nature of the the material for which it generates keys. These protocols have three distinct stages:

1. The programmer-server interaction, where the programmer gets an encryption key from the server.
2. The programmer-agent interaction, where the programmer gives the agent the encrypted message, some data required (but not sufficient) to decrypt the cipher-text, and information as to where to go to get the additional data required to decrypt the cipher-text.
3. The agent-server interaction, where the agent gets the data needed to construct the decryption key and decrypt the cipher-text.

Note that in several cases the first or last aspects are trivial and can be satisfied by publications by the sever.

The *forward-time* constructions permit key generation only after a given time while the *backward-time* permit key generation only before it. These constructions can be nested to permit key generation only during a certain time interval.

The main weakness of these constructions is that the server could collude with an attacker to analyze the agent. This type weakness can easily be abated using the methods discussed in section 6.

4.1 Forward-Time Hash Function

The first time-based construction uses a one-way function, such as SHA-1 [NIST93] or RIPEMD-160 [DBP96], as its sole cryptographic primitive. Let S be a secret belonging to the server.

1. The programmer sends the target time, T^*, and a nonce, R, to the server.
2. The server sets T to the current time and returns to the programmer T and $\mathcal{H}(\mathcal{H}(S, T^*), \mathcal{H}(R, T))$.
3. The programmer sets $P = \mathcal{H}(R, T)$ and $K = \mathcal{H}(\mathcal{H}(S, T^*), \mathcal{H}(R, T))$. The programmer uses K to encrypt the message to the agent, and gives the agent a copy of P. He then lets the agent loose in the world.
4. The agent continuously requests the current time's secret from the server.
5. The server returns $S_i = \mathcal{H}(S, T_i)$. (Alternatively, the server could simply continuously broadcast S_i and the agent could simply watch the broadcast stream.)
6. The agent tries to use $K = \mathcal{H}(S_i, P)$ to decrypt its instructions. It will succeed precisely when $S_i = \mathcal{H}(S, T^*)$ which is when $T_i = T^*$.

This construction has several properties worth listing:

- The use of the current time in the construction of P prevents an analyst from using the server to stage a dictionary attack.
- The form of the daily secret could easily be made hierarchical so that the secret for one day could be used to compute previous daily secrets.
- The use of a nonce, R, reduces the feasibility of a forward time dictionary attack against the server in addition to obscuring the request date of a particular key.
- Should this construction be used maliciously so that the courts order the server to participate in a particular analysis, the server could use P to compute an individual key without giving away all keys for that day.

4.2 Forward-Time Public Key

The second time-based construction uses public-key encryption, such as RSA [RSA78] or ElGamal [ElG84]. For each time T_i, the server has a method of generating a public-key/private-key key pair, (D_i, E_i). The server can either store these key pairs, or regenerate them as required.

1. The programmer sends a target time T^* to the server.
2. The server returns the public key, D^*, for that time.
3. The programmer uses D^* to encrypt the message to the agent. He then lets the agent loose in the world.
4. The agent continuously requests the current time's private key from the server.
5. The server returns E_i. (Again, the server could continuously broadcast E_i).
6. The agent tries to use E_i to decrypt its instructions. It will succeed precisely when $E_i = E^*$, which is when $T_i = T^*$.

This protocol has the advantage that the programmer need not interact with the server in Steps (1) and (2). The server could simply post the D_i values for values of i stretching several years in the future, and the programmer could just choose the one he needs. In this application, the server could be put in a secure location—in orbit on a satellite, for example—and be reasonably safe from compromise.

4.3 Backward-Time Hash Function

The backward time construction also uses one-way functions as its sole cryptographic primitive. Again S is a secret belonging to the server.

1. The programmer sends the target time T^* and a nonce R to the server.
2. The server returns $\mathcal{H}(S, R, T^*)$ if and only if T^* is in the future.
3. The programmer sets K to the returned value and gives the agent a copy of R and T^*.
4. At time T, the agent sends the target time T^* and a nonce R to the server. It will receive the valid key K in return if and only if T^* is later than T.

Backward time constructions in which the target time T^* is unknown to the agent are also possible and are explained in section 6.

5 General Server Constructions

The general server construct uses one-way functions and a symmetric encryption algorithm. Again S is a secret belonging to the server.

1. The programmer sends the server a program P and the hash of a particular possible output $\mathcal{H}(N)$ of the program P.
2. The server returns $E_S(P)$ and $\mathcal{H}(S, P, \mathcal{H}(N))$.
3. The programmer sets $K = \mathcal{H}(S, P, \mathcal{H}(N))$ and uses it to encrypt the message to the agent. The programmer then gives $E_S(P)$ to the agent.
4. The agent gives $E_S(P)$ to the server.
5. The server decrypts the program $P = D_S(E_S(P))$, executes it, and sets M equal to the hash of its output. It then returns $\mathcal{H}(S, P, M)$ to the agent.
6. The agent tries to use the returned value as its key. It will succeed precisely when the output of the run program matches the programmer's original expectations.

his generic construction requires a safe execution environment, as that provided by Java in web browsers, with the additional constraint that the environment does *not* contain the secret S.

While each of the previously discussed constructs can be built using this method, they loose several of the anonymity features, and require explicit agent-server interaction.

6 Further Constructions

These basic constructions can be assembled into higher level constructions.

6.1 Reduced dictionary

One way of making dictionary attacks [Kle90] infeasible is forcing the attacker to use much too large a dictionary. Let $\mathcal{S}$ be a large collection of data and $\mathcal{S}_l \subset \mathcal{S}$ be a much smaller subset of $\mathcal{S}$ determined by the execution environment. Suppose that we know $x_1, \ldots x_n \in \mathcal{S}_l$. Due to the size disparity between $\mathcal{S}$ and $\mathcal{S}_l$ it is feasible to search through all n-tuples in $\mathcal{S}_l$ such that $\mathcal{H}(\cdot, \ldots, \cdot) = \mathcal{H}(x_1, \ldots, x_n)$ while the analogous search in $\mathcal{S}$ is not possible.

A concrete example of this is to let $\mathcal{S}$ be the *complete* collection of canonical DNS names[3] of all hosts and $\mathcal{S}_l$ be the sub-collection of names from hosts inside a domain behind a firewall. Searching through all name triples in $\mathcal{S}_l$ would be quite easy while searching through all triples in $\mathcal{S}$ would be impossible.

This construction would be useful in the virus example of Section 3.3.

6.2 Thresholding

We note that we can easily create a threshold system using the ideas of secret sharing. Suppose that $\mathcal{S}$ is a set of observations of cardinality n and that we wish to be able to construct a key K if m of them are present. We let $T(m, n)$ be a secret sharing scheme with shares $s_1, \ldots, s_n$ for share holders $1, \ldots, n$. Then for each $x_i \in \mathcal{S}$ we tell the agent that share holder i has name $\mathcal{H}(N, x_i)$ and that his share is generated by the function $\mathcal{H}(\cdot) \oplus \mathcal{H}(x_i) \oplus s_i$.

6.3 Nesting

These constructions can be nested: one environmental key can decrypt a section of the agent, which would then yield another encrypted section requiring yet another environmental key. These nestings can be used to create more complex environmental constructions:

For example:

- Forward-time + Backward-time = time interval
- Forward-time + Basic = Forward time, but only if a specific event has occurred.

Agents can slough off previous information and thus can only be analyzed at times of metamorphosis. In other words, after an agent has triggered based on

[3] roughly the full name a host including domain information

an environmental condition, an attacker could not analyze the agent to determine what the condition was. Moreover, the attacker could not tell where the post-transformation agent was a product of the pre-transformation agent. This properties gives rise to many useful anonymity constructions.

7 Conclusions

As applications that allow mobile code become more prevalent, people will want to limit what an attacker can learn about themselves. The notion of clueless agents presented in this paper will have all sorts of applications: blind search engines (patents and product ideas), Manchurian mobile agents, expiration dates by backward-time constructs, intrusion detection systems which are difficult to bypass (they can watch for exploit without revealing nature of the vulnerability they are guarding), logic bombs, directed viruses (both good and bad), remote alarms, etc. The notion of a software construction that hides its true nature is a powerful one, and we expect many other applications to appear as the technology matures.

References

[ATT92] AT&T, Telephone Security Device 3600–User's Manual, AT&T, 1992.

[DOW92] W. Diffie, P.C. can Oorschot, and M.J. Weiner, "Authentication and Authenticated Key Exchanges," *Designs, Codes, and Cryptography*, v. 2, 1992, pp. 107–125.

[DBP96] H. Dobbertin, A. Bosselaers, and B. Preneel, "RIPEMD-160: A Strengthened Version of RIEPMD," *Fast Software Encryption, Third International Workshop,* Springer-Verlag, 1996, pp. 71–82.

[ElG84] T. ElGamal, A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms, IEEE Transactions on Information Theory, IT-31 (1985) 469-472.

[Kle90] D.V. Klein, "Foiling the Cracker: A Security of, and Implications to, Password Security," *Proceedings of the USENIX UNIX Security Workshop*, Aug 1990, pp. 5–14.

[Mye94] E.D. Myers, "STU-III—Multilevel Secure Computer Interface," *Proceedings of the Tenth Annual Computer Security Applications Conference*, IEEE Computer Society Press, 1994, pp. 170–179.

[NIST93] National Institute of Standards and Technology, NIST FIPS PUB 180, "Secure Hash Standard," U.S. Department of Commerce, May 1993.

[RSA78] R. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," *Communications of the ACM*, v. 21, n. 2, Feb 1978, pp. 120-126.

This article was processed using the LaTeX macro package with LLNCS style