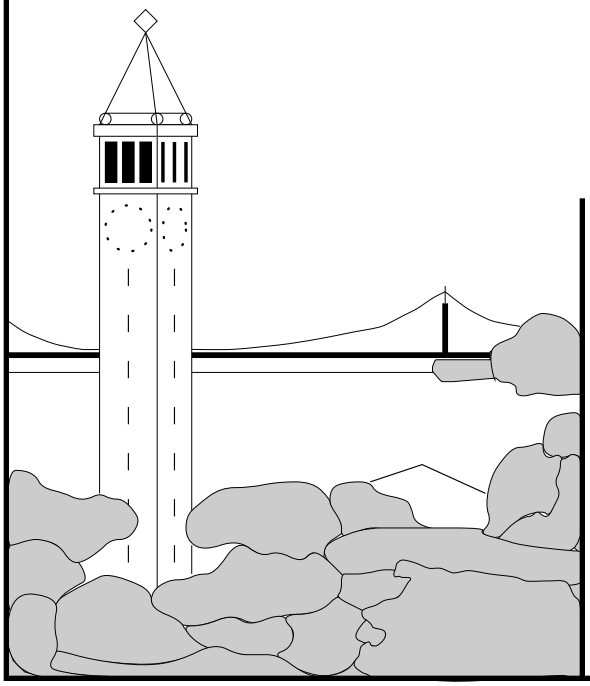
Hidden Markov Model Cryptanalysis

Chris Karlof *David Wagner*



Report No. UCB//CSD-03-1244

Computer Science Division (EECS)
University of California
Berkeley, California 94720

Hidden Markov Model Cryptanalysis

Chris Karlof and David Wagner

Department of Computer Science,
University of California at Berkeley, Berkeley CA 94720, USA,
{ckarlof,daw}@cs.berkeley.edu

Abstract. We present *HMM attacks*, a new type of cryptanalysis based on modeling randomized side channel countermeasures as Hidden Markov Models (HMM's). We also introduce Input Driven Hidden Markov Models (IDHMM's), a generalization of HMM's that provides a powerful and unified cryptanalytic framework for analyzing countermeasures whose operational behavior can be modeled by a probabilistic finite state machine. IDHMM's generalize previous cryptanalyses of randomized side channel countermeasures, and they also often yield better results. We present efficient algorithms for key recovery using IDHMM's. Our methods can take advantage of multiple traces of the side channel and are inherently robust to noisy measurements. Lastly, we apply IDHMM's to analyze two randomized exponentiation algorithms proposed by Oswald and Aigner. We completely recover the secret key using as few as ten traces of the side channel.

1 Introduction

Randomized countermeasures [1–8] for side channel attacks [8–10] are a promising, inexpensive alternative to hardware based countermeasures. In order to gain strong assurance in randomized schemes, we need some way to analyze their security properties, and ideally, we would like general-purpose techniques. To this end, we present *HMM attacks*, a new type of cryptanalysis based on modeling countermeasures as Hidden Markov Models (HMM's) [11]. We also introduce Input Driven Hidden Markov Models (IDHMM's), a generalization of HMM's. IDHMM's are particularly well suited for analyzing any randomized countermeasure whose internal operation can be modeled by a probabilistic finite state machine.

Hidden Markov Models (HMM's) [11] are a well-studied model for finite-state stochastic processes. An execution of an HMM consists of a sequence of hidden, unobserved states and a corresponding sequence of related, observable outputs. HMM's are memoryless: given the current state, the conditional probability distribution for the next state is independent of all previous states. The main problem of interest in HMM's is the *inference problem*, to infer the values of the hidden, unobserved states given only the sequence of observable outputs. The Viterbi algorithm [12] is an efficient dynamic programming algorithm for solving this problem.

At first glance, HMM's seem perfect for analyzing randomized countermeasures which can be modeled by a probabilistic finite state machine: the hidden states of the HMM represent the internal states of the countermeasures and the observable outputs represent observations of the side channel. However, HMM's have deficiencies

Table 1. Summary of attacks on OA1 and OA2, two randomized side channel countermeasures proposed by Oswald and Aigner. Note that our new attacks are the first to work even with a noisy side channel.

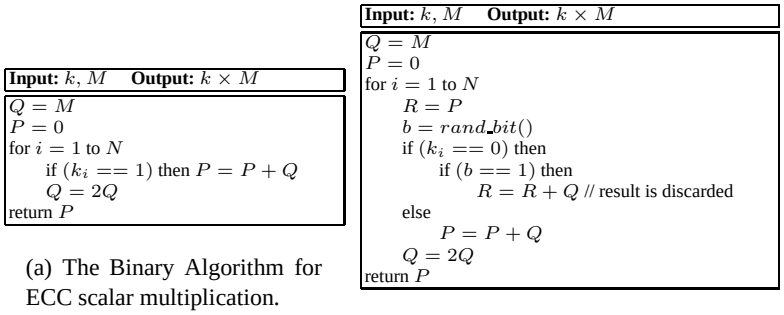
Attack	Relevant countermeasure	Observation error (p_e)	Number of traces needed to recover the secret key	Workfactor
Okeya-Sakurai [13]	OA1	0	292	minimal
C.D. Walter [14]	OA1, OA2	0	2–10	minimal
HMM attacks (new)	OA1, OA2	0	10	minimal
HMM attacks (new)	OA1, OA2	0	5	2^{38}
HMM attacks (new)	OA1, OA2	0.1	10	2^{38}
HMM attacks (new)	OA1, OA2	0.25	50–500	2^{38}

which prevent them from being directly applicable. Firstly, HMM's do not model inputs. HMM's model processes as a sequence of states. However, the internal operation of a randomized countermeasure is likely to be dependent on both the current internal state as well as an input: the secret key. IDHMM's extend HMM's to handle inputs so we can accurately model randomized keyed countermeasures. Secondly, standard inference techniques like Viterbi's algorithm cannot leverage multiple output traces of an HMM. However, the ability to handle multiple output traces will make our key recovery attacks more powerful. To address this, we present an efficient approximate inference algorithm for IDHMM's that handles multiple output traces.

To demonstrate how HMM attacks can be used in practice, we show how to break two randomized exponentiation algorithms proposed by Oswald and Aigner [2]. Previously known attacks [13, 14] against these algorithms assume the ability to perfectly distinguish between elliptic curve point additions and doublings in the side channel. We present more powerful attacks which are robust to noise. A summary of our attacks in comparison to previous work is shown in Table 1.

2 Modeling Randomized Side Channel Countermeasures as Probabilistic Finite State Machines

Many authors have proposed randomization as a way to limit the security risks from information leaked over side channels [1–8, 15]. However, the security afforded by randomization in this setting is not clear. Side channel attacks are typically successful because of the high correlation between the information leaked over the side channel and the internal state of the device, most notably a secret key used in various cryptographic operations. The hope behind randomized countermeasures is that the side channel information will become randomized as well, thus making it harder to analyze. An ideal randomized countermeasure would completely disassociate the side channel information from the internal state of the device, or more formally, for any set of measurements of the side channel, the likelihood of an adversary guessing any information about the internal state of the device would be the same as if the adversary had observed no side channel information at all. Some examples of randomized countermeasures include randomized exponentiation algorithms [1–4], random window methods [5, 6], randomized



```
Input: k, M    Output: k × M
Q = M
P = 0
for i = 1 to N
    if (k_i == 1) then P = P + Q
    Q = 2Q
return P
```

(a) The Binary Algorithm for ECC scalar multiplication.

```
Input: k, M    Output: k × M
Q = M
P = 0
for i = 1 to N
    R = P
    b = rand_bit()
    if (k_i == 0) then
        if (b == 1) then
            R = R + Q // result is discarded
    else
        P = P + Q
    Q = 2Q
return P
```

(b) A randomized variant of the Binary Algorithm for ECC multiplication.

Fig. 1. Introducing randomness into the Binary Algorithm for ECC scalar multiplication.

instruction execution [7], randomized timing shifts [8], randomized blinding of the secret key [15], and randomized projective coordinates [15].

We introduce a new cryptanalytic technique based on Hidden Markov Models to analyze such randomized countermeasures. To help give the intuition behind our analysis, we first give a simple example of a fabricated countermeasure that uses randomization, show how to model its operation using a probabilistic finite state machine, and then motivate the use of Hidden Markov Models to analyze its security.

2.1 A Simple Randomized Countermeasure

Consider the randomized variant of the standard binary algorithm for doing scalar multiplications over elliptic curves shown in Figure 1(b). Assume $k = k_N k_{N-1} \ldots k_2 k_1$ is the N bit secret key and M and P are points on the elliptic curve.

The major difference between the algorithm in Figure 1(b) and the standard Binary Algorithm is as follows: in each iteration, if the next key bit is 0, then with probability $1/2$ our algorithm will execute a discarded spurious addition, but if the next key bit is 1, it behaves the same as the standard Binary Algorithm. This randomized variant of the Binary Algorithm is completely artificial and by no means secure. It was created solely to demonstrate how randomness might be used in the construction of side channel countermeasures and will serve as a running example to illustrate our techniques.

Now, assume that it is possible for an adversary observing the side channel to distinguish between elliptic curve point additions and elliptic curve point doublings in a single scalar multiplication. Then the adversary's observation of a single scalar multiplication can be represented as a sequence $(y_1, y_2, \ldots, y_N),\ y_i \in \{D, AD\}$, where D represents an elliptic curve point doubling and A represents an elliptic curve point addition. Each y_i represents the operations observed during the processing of a single bit of the key. We refer to such a sequence as a *trace*.

Note there is no longer a one-to-one correspondence between each possible trace and each possible key. Rather, each given sequence of observable operations is consis-

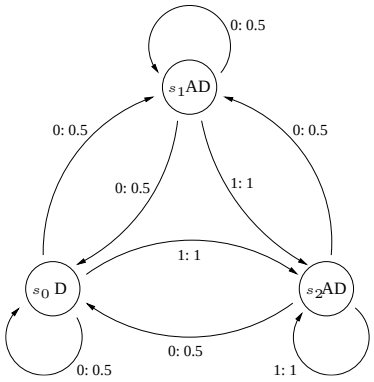


Fig. 2. A probabilistic finite state machine that models the operation of the randomized exponentiation algorithm in Figure 1(b).

tent with several possible keys. For example, if the trace from a scalar multiplication using the algorithm in Figure 1(b) is (AD, AD, D), then there are four possible keys consistent with this trace: namely, 011, 001, 010, or 000.

2.2 Probabilistic Finite State Machines

Although there are clearly many ad-hoc ways to break the algorithm in Figure 1(b), its primary purpose is to illustrate the development of a general technique for analyzing randomized countermeasures. Several weaknesses have been discovered in some existing randomized countermeasures [13, 14, 16], but the analysis techniques used are often specific to the particular countermeasure, and it is not obvious how to generalize them to a framework applicable to a larger class of algorithms. The primary benefit of a general analytical framework is that it enables the analysis of a large class of randomized countermeasures while minimizing the overhead needed to analyze any one particular algorithm. Although such a framework by itself may not in general be able to prove the security of every conceivable countermeasure, it can help quickly determine if a countermeasure is insecure, potentially saving a cryptanalyst many hours of work.

A key component for a general analytical framework is a good operational model of the countermeasures. A simple model applicable to many randomized countermeasures is a *probabilistic finite state machine*. The resulting finite state model for our running example can be easily constructed from its algorithmic description and is shown in Figure 2. Each state corresponds to a full iteration of the loop in Figure 1(b) (i.e, the processing of one key bit in its entirety) and is labeled with the operations (D or AD) that may be observed when that state is visited. Each edge is annotated with a bit from the key and a probability. In general, one of the states would be designated as initial, but in this example any state can serve as the initial state. The model of execution is simple: given the current state q_i and the next bit of the key k_{i+1}, the next state q_{i+1} is determined probabilistically according to the probabilities on those outgoing edges of q_i that are annotated with k_{i+1}.

While the edges capture the control structure of the algorithm, the states abstract away the details of the calculation. However, the label on each state indicates the observable information that leaks through the side channel when the process enters the state. In this example, the observations are what type of operations (elliptic curve point additions and/or doublings) are executed while in a particular state. Note however, since the state machine is randomized, some particular traces could arise from several different paths through the machine. For example, the trace (AD, AD, D) could arise from any of the following paths through the state machine in Figure 2: $s_2s_2s_0$, $s_2s_1s_0$, $s_1s_2s_0$, or $s_1s_1s_0$.

More formally, we define a probabilistic finite state machine to be a sextuple

$$M = (S, I, \delta, O, s_0, \mu)$$

where

S is a finite set of internal states,

I is a finite set of input symbols,

$\delta : S \times S \times I \rightarrow [0, 1]$ is a function called the transition function,

O is a finite set of symbols that represent operations observable over the side channel,

$s_0 \in S$ is the initial state,

$\mu : S \rightarrow O$ is a function associating an observable operation with every state,

and the following condition is satisfied:

$$\forall s_i \in S, \forall b \in I, \sum_{s_j \in S} \delta(s_i, s_j, b) = 1 \; .$$

In our setting, the set of input symbols is $I = \{0, 1\}$, representing the bits of a secret key.

For a key $k = k_N k_{N-1} \ldots k_2 k_1$, define an *execution* q of $M = (S, \delta, O, s_0, \mu)$ on k to be a sequence $q = (q_0, q_1, \ldots, q_{N-1}, q_N)$, where $q_0 = s_0$ and $q_i \in S$, such that for $0 \leq i < n$, $\delta(q_i, q_{i+1}, k_{i+1}) > 0$. Define a *trace* y of an execution q to be a sequence $y = (y_1, y_2, \ldots, y_N)$, where $y_i = \mu(q_i), \forall i \neq 0$. In the case of our example, an execution corresponds to the sequence of internal states traversed during a scalar multiplication of the secret key k, and the trace of that execution represents the sequence of observable elliptic curve point additions and doublings.

2.3 The Key Recovery Problem for Probabilistic Finite State Machines

Since one of the primary goals of side channel attacks is to recover the secret key stored within a target device, we wish to solve the following problem:

> KEY RECOVERY PROBLEM FOR PROBABILISTIC FINITE STATE MACHINES
>
> Let M be a probabilistic finite state machine. Generate a random N bit key k and an execution q of M on k. Let y be the trace of q. The Key Recovery Problem for probabilistic finite state machines is to find k given M and y.

One approach to solving the Key Recovery Problem for probabilistic finite state machines is the following: 1) Given a trace y and machine M, try to infer the execution q it resulted from, and then 2) infer k from q. Step 2 becomes easy if we restrict M to be *faithful*. A probabilistic finite state state machine $M = (S, \delta, O, s_0, \mu)$ is said to be *faithful* if it satisfies the following property:

$$\forall s_i, s_j \in S, \text{ if } \delta(s_i, s_j, 0) > 0, \text{ then } \delta(s_i, s_j, 1) = 0 \, .$$

For faithful machines, there is a one-to-one correspondence between an execution q and the key k used in that execution. This is because for every pair of consecutive states s_i, s_j in an execution, there is no ambiguity in what bit annotated the corresponding directed edge that was taken from s_i to s_j. Note that this condition does not limit the expressiveness of our framework by restricting M in any significant way. If for a machine M there exists s_i, s_j such that $\delta_M(s_i, s_j, 0) > 0$ and $\delta_M(s_i, s_j, 1) > 0$, then an observationally equivalent machine M' can be constructed that is identical to M except state s_j is replaced by two states s_{j_1}, s_{j_2} such that $\delta_{M'}(s_i, s_{j_1}, 0) = \delta_M(s_i, s_j, 0)$, $\delta_{M'}(s_i, s_{j_1}, 1) = 0$, $\delta_{M'}(s_i, s_{j_2}, 0) = 0$, and $\delta_{M'}(s_i, s_{j_2}, 1) = \delta_M(s_i, s_j, 1)$. Thus, without loss of generality, we will only consider probabilistic finite state machines that are faithful.

2.4 The State Inference Problem for Probabilistic Finite State Machines

We define the State Inference Problem for probabilistic finite state machines as follows:

> STATE INFERENCE PROBLEM FOR PROBABILISTIC FINITE STATE MACHINES
>
> Let M be a probabilistic finite state machine. Generate a random N bit key k and an execution q of M on k. Let y be the trace of q. The State Inference Problem for probabilistic finite state machines is to find q given M and y.

Because of the one-to-one correspondence between q and the key k used in that execution, solving the State Inference Problem for M and y is equivalent to solving the Key Recovery Problem for M and y[1].

One way an adversary might try to solve the State Inference Problem for $M = (S, I, \delta, O, s_0, \mu)$ and y of length N is to treat the unknown execution q as a random variable Q with sample space S^{N+1} and use *maximum likelihood decoding*. A simple implementation of maximum likelihood decoding involves two steps:

Input: trace y, machine M
1. Calculate $\mathbf{Pr}[Q = s|y], \text{ for each } s \in S^{N+1}$.
2. Output $q = \underset{s \in S^{N+1}}{\operatorname{argmax}} \mathbf{Pr}[Q = s|y]$.

The adversary's output is the most likely execution q for the given trace y, which then yields the most likely key k.

[1] Although we have formulated both problems in a way that implies deterministic solutions, randomized algorithms with significant success probability are acceptable as well.

A naive implementation of step 1 will have a running time exponential in the length of the trace. However, we will see how to transform a probabilistic finite state machine into a Hidden Markov Model, in which there is an equivalent State Inference Problem with a polynomial running time solution. In addition to having efficient inference algorithms, we will see that Hidden Markov Models have other advantages as well.

In this section, we introduced probabilistic finite state machines, an intuitive technique for modeling the operation of randomized countermeasures, and we demonstrated the use of the model with an artificial yet instructive example. In the remainder of this paper, we will show how Hidden Markov Models not only provide a sound, well-studied framework, but also how they can be extended into even more powerful and flexible cryptanalytical tools for analyzing randomized countermeasures.

3 Assumptions

Before formally describing our analytical framework for randomized countermeasures using Hidden Markov Models, we will make our assumptions more precise. Our analysis depends on the following assumptions:

- We have collected a set of L traces from the side channel, corresponding to L executions of the countermeasure, all using the same secret key.
- Each trace of the side channel can be uniquely written as $(y_1, y_2, \ldots, y_N)$ where each y_i is an element of some finite observation set O. In the example presented in Section 2, $O = \{D, AD\}$.
- The operations in O can be probabilistically distinguished from each other.
- Each observation y_i from O can be associated with the processing of a single key bit position, and vice versa.

If the attacker is lucky, the side channel reveals exactly which action from O has been taken, and thus the observation traces $(y_1, y_2, \ldots, y_N)$ are free of errors. This is the model some previous work has used, and it does simplify analysis. However, this assumption may not always be realistic.

In the more general case, observations may only yield partial information on the actual trace, hence our measurements may contain errors. As we will see in Section 4, our techniques are still applicable in this setting. When there are only two different types of observations, a simple model of this behavior is that each observation has probability $1 - p_e$ of being correct and probability p_e of being mischaracterized. This setting may be more realistic in practice, particularly for devices that try to make all operations look alike.

4 Input Driven Hidden Markov Models as a Model for Randomized Side Channel Countermeasures

In Section 2, we outlined an approach for analyzing randomized countermeasures which infers the most likely secret key from the sequence of observable operations. However, this approach is not only intractable, but has other deficiencies as well. Four main challenges remain:

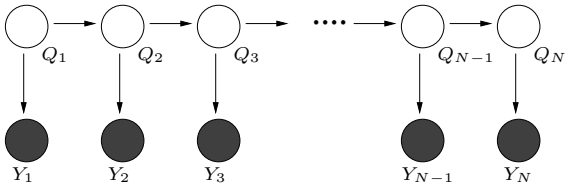


Fig. 3. An execution of a Hidden Markov Model, represented in a probabilistic graphical model. This figure depicts one execution of the HMM. Each node represents a random variable, and the directed edges indicate conditional dependencies. A shaded node indicates the corresponding variable is observed (i.e. outputs we can observe), while unshaded nodes are unobserved (i.e. what we wish to recover).

1. **Efficient inference algorithms are needed.** A naive implementation of maximum likelihood decoding for a single trace has running time exponential in the length of the trace. In order to be useful, inference algorithms must scale better.
2. **Side channel measurements may be noisy.** As we mentioned in Section 3, our measurements of the side channel may be noisy and contain errors. It is desirable to have techniques that tolerate noise.
3. **We need a model that handles inputs.** Hidden Markov Models will serve as a starting point for our techniques, but HMM's only have outputs and do not model inputs. In order to accurately model the secret key, we need a framework that models processes with both inputs and outputs.
4. **One trace is typically not enough.** For any reasonable countermeasure, the set of possible keys consistent with a single trace will be large. Hence, attacks that examine only a single trace are unlikely to be successful. However, by gathering multiple traces that result from use of the same key, we may be able to narrow the list of likely candidates. Thus, it is desirable to have techniques that can analyze an arbitrary number of traces. This will make our analysis both more general and more powerful.

First, we will show how Hidden Markov Models can be used to solve problems 1 and 2. Then, we introduce an extension to HMM's, *Input Driven Hidden Markov Models*, that address problems 3 and 4.

4.1 Hidden Markov Models

Hidden Markov Models (HMM's) [11] are a well-studied method for modeling finite-state stochastic processes. The word "hidden" indicates that the states of the process are not directly observable. Instead, related to each state is an output which is observable. One of the main problems of interest for Hidden Markov Models is the inference problem: to infer the most likely sequence of values for the hidden states given only the observations. Since there exist efficient algorithms for solving the inference problem in HMM's, this motivates trying to model randomized countermeasures as HMM's in a

way so that the key recovery problem for a randomized countermeasure becomes the inference problem in an HMM.

HMM's induce two sequences of finite random variables: the hidden states, Q_1, Q_2, ..., Q_N, and the observations[2], $Y_1, Y_2, \ldots, Y_N$. Like regular Markov Models, the value of the next state is dependent only on the current state and not any previous states. That is, the distribution of Q_n is conditionally independent of $Q_1, Q_2, \ldots, Q_{n-2}$ given Q_{n-1}. In addition, it is assumed that the distribution of Y_n is conditionally independent of everything else given Q_n.

HMM's are parameterized by the local conditional distributions $\mathbf{Pr}[Q_n|Q_{n-1}]$ and $\mathbf{Pr}[Y_n|Q_n]$, both of which are assumed to be independent of n. If $S = \{s_1, s_2, \ldots, s_M\}$, the conditional distribution $\mathbf{Pr}[Q_n|Q_{n-1}]$ is parameterized by a $M \times M$ transition matrix A, where $A_{ij} = \mathbf{Pr}[Q_n = s_j|Q_{n-1} = s_i]$. Since in our setting the sample space of the observations Y_n is a finite observation set $O = \{o_1, o_2, \ldots, o_J\}$, the conditional distribution $\mathbf{Pr}[Y_n|Q_n]$ is parameterized by a $M \times J$ output matrix B, where $B_{ij} = \mathbf{Pr}[Y_n = o_j|Q_n = s_i]$.

In summary, for our setting, a Hidden Markov Model is defined by the quintuple

$$H = (S, O, A, B, s_0)$$

where

S is a finite set of internal states,
O is a finite set of symbols that represent operations observable over the side channel,
A is a $|S| \times |S|$ matrix where $A_{ij} = \mathbf{Pr}[Q_n = s_j|Q_{n-1} = s_i]$,
B is a $|S| \times |O|$ matrix where $B_{ij} = \mathbf{Pr}[Y_n = o_j|Q_n = s_i]$,
$s_0 \in S$ is the initial state.

We refer to a realization $q_1, q_2, \ldots, q_N$ of the random variables $Q_1, Q_2, \ldots, Q_N$ as an *execution* of the HMM, and to a realization $y_1, y_2, \ldots, y_N$ of the random variables $Y_1, Y_2, \ldots, Y_N$ as a *trace* of the HMM. Recall that traces are observable whereas executions are generally not.

Hidden Markov Models have a graphical representation as the probabilistic graphical model [11] shown in Figure 3. Probabilistic graphical models are a graphical representation of a set of random variables where each node represents a random variable and the directed edges indicate conditional dependencies. A shaded node indicates the corresponding variable is observed, while unshaded nodes are unobserved. In the case of HMM's, the shaded nodes correspond to the observations Y_n, and the unshaded nodes correspond to the hidden, unobserved states Q_n.

Consider a probabilistic finite state machine with no inputs, i.e., with no keys. That is, for $M = (S, \delta, O, s_o, \mu)$, the transition function is given by $\delta : S \times S \to [0, 1]$ rather than $\delta : S \times S \times I \to [0, 1]$. If we wish to infer the most likely execution for M given a trace y of length N, we can construct a Hidden Markov Model $H = (S, O, A, B, s_0)$ where the unknown states in the execution can be modeled by the random variables $Q_1, Q_2, \ldots, Q_N$ and the corresponding observations are realizations of $Y_1, Y_2, \ldots, Y_N$.

[2] HMM's can handle real-valued observations as well.

The transition matrix A can be easily constructed from δ, and the output matrix B is completely deterministic, i.e., each row of B has one 1 and $(|O| - 1)$ 0's.

Solving the State Inference Problem for M and y reduces to solving a similar state inference problem for H and y. The State Inference Problem for a Hidden Markov Model H given a trace $y = (y_1, y_2, \ldots, y_n)$ is as follows:

> STATE INFERENCE PROBLEM FOR HIDDEN MARKOV MODELS
>
> Let H be Hidden Markov Model. Generate an execution q of H and let y be a trace of q. The State Inference Problem for Hidden Markov Models is find q given H and y.

The standard approach for finding q in the State Inference Problem for HMM's is maximum likelihood decoding. However, as we have seen before, naive implementations for finding

$$q = \underset{s \in S^N}{\operatorname{argmax}} \mathbf{Pr}[Q = s | Y = y]$$

will have running time exponential in N. However, the Viterbi algorithm [12] is a well-known dynamic programming solution to the State Inference Problem for HMM's with running time $O(|S|^2 \cdot N)$. This addresses the first challenge, the need for efficient inference algorithms.

HMM's can also address the second problem of noisy side channel measurements. This can be handled by proper parameterization of the distribution $\mathbf{Pr}[Y_n|Q_n]$. For example, in the toy example presented in section 2, for perfect observations, $\mathbf{Pr}[Y_n = AD|Q_n = s_2] = 1$. Noisy observations can be modeled by assuming observations are only probabilistically correct, e.g., $\mathbf{Pr}[Y_n = AD|Q_n = s_2] = 0.7$ and $\mathbf{Pr}[Y_n = D|Q_n = s_2] = 0.3$.

4.2 Input Driven Hidden Markov Models

HMM's are not completely adequate for modeling most countermeasures. In most countermeasures, the next state depends not only on the current state, but also on the next bit of the key. In the context of HMM's, we would like the key to serve as a sort of input to the HMM. However, HMM's unfortunately do not have inputs. Therefore, we extend the notion of HMM's to include the possibility of inputs by introducing *Input Driven Hidden Markov Models* (IDHMM's).

IDHMM's extend HMM's in two fundamental ways. First, the unknown input is treated as a random variable $K = (K_1, K_2, \ldots, K_N)$ such that K_n is input to the underlying HMM at step n. The local conditional distribution are updated to reflect this, i.e., we replace $\mathbf{Pr}[Q_n|Q_{n-1}]$ with $\mathbf{Pr}[Q_n|Q_{n-1}, K_n]$. Second, since one of the motivations behind developing IDHMM's was to analyze multiple traces, we need to add additional random variables to model additional execution/trace pairs. Thus, $Y^1, Y^2, \ldots, Y^L$ will represent a list of L traces, where $Y^l = (Y_1^l, Y_2^l, \ldots, Y_N^l)$. Also, $Q^1, Q^2, \ldots, Q^L$ will represent the corresponding L sequences of hidden states, where $Q^l = (Q_1^l, Q_2^l, \ldots, Q_N^l)$. IDHMM's assume the same input is used in every execution, which corresponds exactly to the assumption that the same key is used in every execution of the countermeasure.

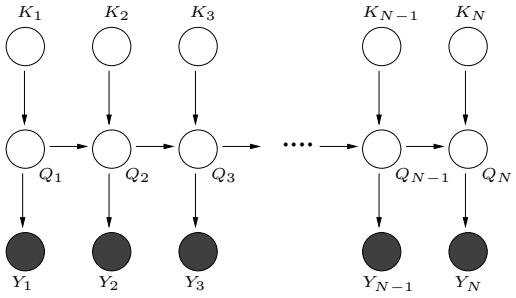


Fig. 4. Input Driven Hidden Markov Models. This figure depicts one execution of an IDHMM on input $K_1, K_2, \ldots, K_N$.

The graphical model shown in Figure 4 represents a single execution of an IDHMM. The input was applied a single time to a produce a single output trace. Figure 5 shows a graphical model representing L traces from L executions of an IDHMM in which the same input is applied in each execution, for some $L > 1$.

An Input Driven Hidden Markov Model is defined by the septuple

$$H = (S, I, O, A, B, C, s_0)$$

where

S is a finite set of internal states,

I is a finite set of input symbols,

O is a finite set of symbols that represent operations observable over the side channel,

A is a $|S| \times |I| \times |S|$ matrix where $A_{ijk} = \mathbf{Pr}[Q_n^l = s_k | Q_{n-1}^l = s_i, K_n = k_j]$,

B is a $|S| \times |O|$ matrix where $B_{ij} = \mathbf{Pr}[Y_n^l = o_j | Q_n^l = s_i]$,

C is a $N \times |I|$ matrix representing the prior distributions for $(K_1, K_2, \ldots, K_N)$,
where $C_{jk} = \mathbf{Pr}[K_j = i_k]$,

$s_0 \in S$ is the initial state.

Since in our setting the input is a binary key chosen uniformly at random, the set of input symbols is $I = \{0, 1\}$ and the prior distributions are $\mathbf{Pr}[K_n = 0] = \mathbf{Pr}[K_n = 1] = 0.5$.

Our final goal is the inference problem for IDHMM's: we want to infer the input key K rather than the sequences of hidden states Q. We define the Key Inference Problem for Input Driven Hidden Markov Model as follows:

KEY INFERENCE PROBLEM FOR INPUT DRIVEN HIDDEN MARKOV MODELS

Let H be an Input Driven Hidden Markov Model. Generate a N bit random input key k and L executions $q = (q^1, q^2, \ldots, q^L)$ of H on k. Let $y = (y^1, y^2, \ldots, y^L)$ be the corresponding L traces. The Key Inference Problem for Input Driven Hidden Markov Models is to find k given H and y.

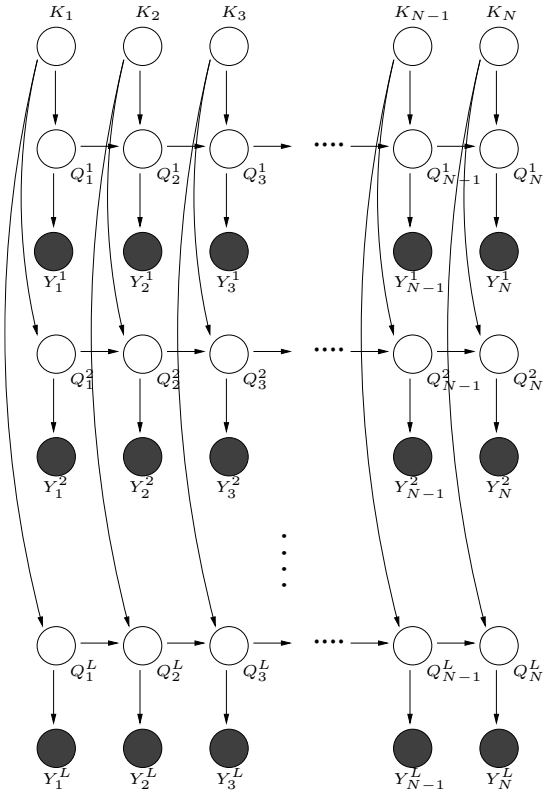


Fig. 5. Modeling Multiple Executions of an Input Driven Hidden Markov Model. This figure depicts L executions of an IDHMM on input $K_1, K_2, \ldots, K_N$.

Ideally, we would like to compute

$$k = \underset{k \in \{0,1\}^N}{\operatorname{argmax}} \mathbf{Pr}[K = k | Y = (y^1, y^2, \ldots, y^L)] \; . \qquad (*)$$

However, we do not know how to compute this efficiently. Therefore, we introduce an approximation: we infer the posterior probabilities for each bit of the key separately, and then we use the most likely value of each bit to infer the entire key. This amounts to computing

$$k = (k_N, k_{N-1}, \ldots, k_2, k_1), \text{ where } k_n = \underset{b \in \{0,1\}}{\operatorname{argmax}} \mathbf{Pr}[K_n = b | Y = (y^1, y^2, \ldots, y^L)] \; .$$

However, even this was too hard for us. Our first attempts at an algorithm to calculate $\mathbf{Pr}[K_n|y]$ using dynamic programming in a manner similar to that in the inference algorithm for HMM's encountered a significant problem: the resulting algorithm had running time exponential in L, the number of traces. Since our goal is to scale with the number of traces, this is unacceptable.

SINGLETRACEINFERENCE(H, D, y'):

Input: An IDHMM H, a distribution D, assumed $\mathbf{Pr}[K = k] = D(k)$, and a trace $y' = (y'_1, y'_2, \ldots, y'_N)$

Output: A distribution D', where $D'(k) = \mathbf{Pr}[K = k | Y' = y']$, using D as our priors on K

1) Use a modified version of the Viterbi algorithm to compute $D'(k) := \mathbf{Pr}[K = k | Y' = y']$, assuming $D(k) = \mathbf{Pr}[K = k]$. Refer to Appendix A for details.

MULTITRACEINFERENCE(H, y):

Input: An IDHMM H and a set $y = (y^1, y^2, \ldots, y^L)$ of L traces

Output: D_L, an approximation to the distribution $\mathbf{Pr}[K = k | Y = y]$

1) Let $D_0 :=$ the uniform distribution on $\{0, 1\}^N$.
2) **for** $i := 1, 2, \ldots, L$ **do**
   $D_i :=$ SINGLETRACEINFERENCE(H, D_{i-1}, y^i)
3) Output D_L.

INFER(H, y):

Input: An IDHMM H and a set $y = (y^1, y^2, \ldots, y^L)$ of L traces

Output: k, a guess at the key

1) Let $\mathbf{Pr}[K_i | Y = y]$ be as given by MULTITRACEINFERENCE(H, y).
2) **for** $i := 1, 2, \ldots, N$ **do**
   if $\mathbf{Pr}[K_i = 1 | Y = y] > 0.5$ **then** $k_i := 1$ **else** $k_i := 0$
3) Output $k_{\text{guess}} = k_N k_{N-1} \ldots k_2 k_1$.

Fig. 6. An approximate inference algorithm using belief propagation. Given a set of traces $y = (y^1, y^2, \ldots, y^L)$ of an Input Driven Hidden Markov Model H, we compute a guess $k_{\text{guess}} =$ INFER(H, y) at the key.

To deal with these challenges, we introduce a new technique based on *belief propagation*. The key idea is to separate L executions of an IDHMM on the same input into L executions of an IDHMM where there are no assumptions about the input used in each execution. In terms of the graphical models, this corresponds to transforming Figure 5 into L copies of Figure 4. We can derive an efficient exact inference algorithm for a single execution of an IDHMM with running time $O(|S|^2 \cdot N)$. By applying this exact inference algorithm separately to each of the L executions, we obtain an algorithm with final running time of $O(|S|^2 \cdot N \cdot L)$.

The problem with this approach is that we are not taking advantage of the fact that the executions all use the same key. Using L traces as input, this approach will output L separate inferences of the key, each derived independently of the others. We can link them by using belief propagation: instead of using the uniform distribution as our prior $\mathbf{Pr}[K_n]$ for each key bit in the L analyses, we use the posterior distribution $\mathbf{Pr}[K_n | y^l]$ calculated from analysis of the l-th trace as the prior distributions $\mathbf{Pr}[K_n]$ while analyzing the $l+1$-st trace. Hence, we propagate any biases on the key bits learned from the l-th trace to the analysis of the $l+1$-st trace. A detailed description of our belief propagation algorithm for inferring the secret key from L traces of an IDHMM is shown in Figure 6. Although the output of this algorithm is only an approximation to what we ideally want in (*), we have found that it works well in practice.

5 Application to Randomized Addition-Subtraction Chains

Oswald and Aigner have proposed two randomized exponentiation algorithms [2] for scalar multiplication in ECC implementations. These algorithms are based on the randomization of addition-subtraction chains. For example, instead of the usual binary decomposition $15P = 8P + 4P + 2P + 1$, $15P$ can alternatively be calculated as as

$$15P = 16P - P = 2(2(2(2(P)))) - P.$$

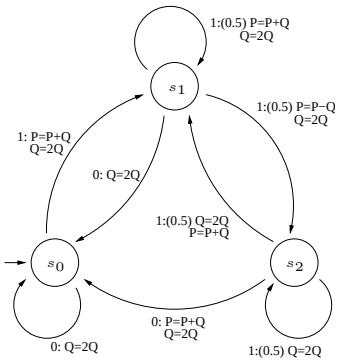
More generally, a series of more than two 1's in the binary representation of k can be replaced by a block of 0's and a -1, i.e., $01^a \mapsto 10^{a-1}\bar{1}$ where $\bar{1}$ represents -1. A second transformation noted by Oswald and Aigner treats isolated 0's inside a block of 1's, i.e., $01^a01^b \mapsto 10^a\bar{1}0^{b-1}\bar{1}$.

Both of these transformations can be modeled by deterministic finite state machines. Oswald and Aigner construct two randomized exponentiation algorithms by introducing randomness into these state machines while still preserving the end semantics of the two transformations. At each step where a transformation may apply, we flip a coin to decide whether or not to apply that transformation. We refer to the randomized construction based on the transformation $01^a \mapsto 10^{a-1}\bar{1}$ as OA1 and the randomized construction based on the transformation $01^a01^b \mapsto 10^a\bar{1}0^{b-1}\bar{1}$ as OA2. The randomized state machine describing the operation of OA1 (as it appears in [2]) is shown in Figure 7(a).

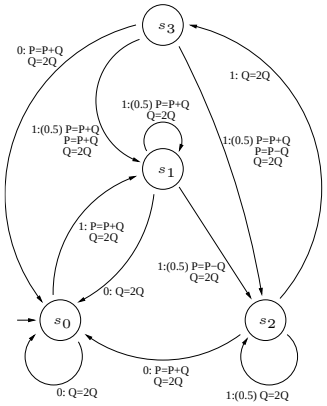
The randomized state machines in [2] that describe the operation of OA1 and OA2 do not conform to our definition of probabilistic state machines in Section 2, but this is easily remedied. The first hurdle is that traces cannot be parsed uniquely as words in $\{D, AD\}^*$. Although it would be convenient if our observable alphabet was $O = \{D, AD\}$, the transition from s_2 to s_1 executes a doubling first and then an addition, resulting in a DA output symbol corresponding to that key bit. This is undesirable because traces fail to be uniquely decodeable: for example, $DADD$ could be interpreted as either (DA, D, D) or (D, AD, D). We remedy this problem by interpreting the automaton in Figure 7(a) slightly differently. We relabel the DA transition from s_2 to s_1 to simply a D (i.e., $Q = 2Q$) and now associate the "owed" addition with each outgoing transition from state s_1. Our output alphabet becomes $O = \{D, AD, AAD\}$, and then each sequence of D and A operations can deterministically be decomposed into a sequence of symbols from O. The resulting state machine is shown in Figure 7(b).

A second hurdle is that Oswald and Aigner place observable operations on the edges, rather than on the states. Fortunately, edge-annotated state machines can easily be transformed into a semantically equivalent state-annotated machine (of the type defined in Section 2) by treating each edge in Figure 7(b) as a state in the probabilistic FSA. This yields a faithful probabilistic finite state machine to which our algorithms can be applied. See Figure 7(c) for the result of this process applied to OA1.

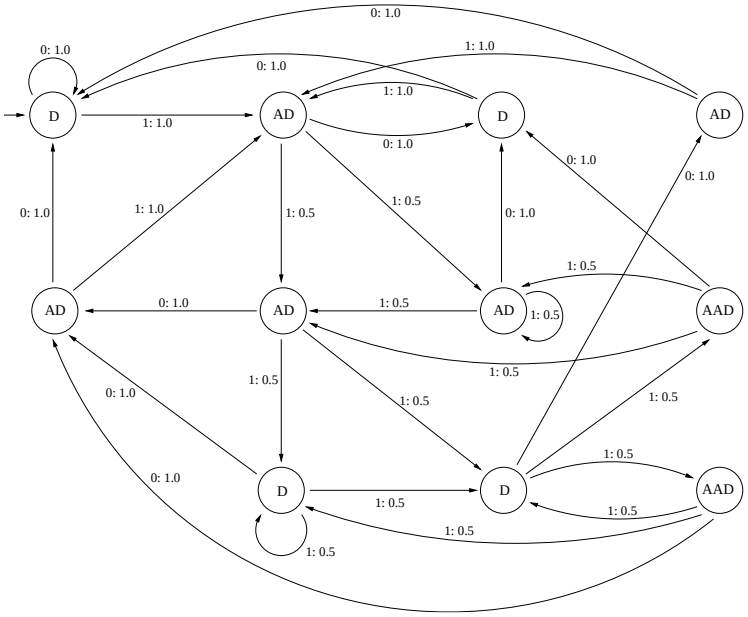
Once we have probabilistic finite state machine representations for the countermeasures, applying our techniques is straightforward. We simulated the operation of both exponentiation algorithms in software. First, we generated a random 192 bit key k. Using k, we then generated a set of traces $y = (y^1, y^2, \ldots, y^L)$. We introduced errors in the traces consistent with observation error p_e. With probability $1 - p_e$, each output



(a) A randomized state machine that represents random application of the transformation $01^a \mapsto 10^a\overline{1}$ of a key k in the scalar multiplication $k \times M$. Q is initialized to M.



(b) A reinterpretation of Figure 7(a) such that the observable operation labeling each edge is a member of $O = \{D, AD, AAD\}$.



(c) A faithful probabilistic state machine (conforming to the definition in Section 2) that models the behavior of Figure 7(b).

Fig. 7. The first Oswald-Aigner construction (OA1).

symbol is observed correctly, and with probability p_e, it is changed to some other output symbol (chosen randomly). We assumed the error probability p_e is known to the attacker, and we incorporated it into the output distribution (i.e., $\mathbf{Pr}[Y_n \mid Q_n]$) of the resulting IDHMM. Treating the OA1 or OA2 countermeasure as an IDHMM driven by k, we then applied the INFER algorithm from Figure 6 to compute

$$k_{\text{guess}} = (k_N, k_{N-1}, \ldots, k_2, k_1), \text{ where}$$
$$k_n = \underset{b \in \{0,1\}}{\operatorname{argmax}} \mathbf{Pr}[K_n = b \mid Y = (y^1, y^2, \ldots, y^L)] \; .$$

The following table summarizes the results of our attacks against OA1 and OA2:

Number of key bits correctly recovered

Countermeasure	p_e	Number of traces used						
		1	5	10	25	50	100	500
OA1	0	170	187	192	192	192	192	192
OA1	0.1	157	178	184	185	187	192	192
OA1	0.25	143	163	173	180	182	183	184
OA1	0.4	120	147	159	168	172	173	174
OA2	0	165	188	192	192	192	192	192
OA2	0.1	156	174	184	187	189	192	192
OA2	0.25	135	161	174	177	180	181	182
OA2	0.4	126	146	154	168	171	172	173

Each entry in the table specifies the number of key bits (out of 192) that we correctly recovered using the corresponding number of traces and given observation error p_e.

Both OA1 and OA2 are clearly insecure under our assumptions in Section 3. With a perfect side channel ($p_e = 0$), we recovered the entire secret key perfectly with as few as 10 traces; also, our techniques remain effective in the presence of noise.

One can also reduce the number of traces by combining our attack with a semi-exhaustive attack over the most likely key candidates. It suffices to recover 182 of the 192 key bits correctly, on average; then we can apply a meet-in-the-middle search over all possible 10-bit error patterns to identify the correct private key, using 2^{38} work[3]. Hence, with a 0.1 probability of observation error, the entire key can be recovered with only 10 traces, and for a 0.25 error probability, the data complexity increases to 50-500 traces.

[3] Suppose we know elliptic curve points x, y, related by $y = kx$, where k is the private key. Assume we know k', where $k' = k \oplus e$ and e is an unknown 10-bit error pattern. Break these 192-bit values into two 96-bit chunks, so $k = k_2 k_1$, etc. Then $y - 2^{96}(k'_2 \oplus e_2)x = (k_1 \oplus e_1)x$, hence we can use a meet-in-the-middle attack. A fraction $(\binom{96}{4} + \binom{96}{5} + \binom{96}{6})/2^{10} \approx 0.66$ of all 10-bit error patterns split so that both halves have hamming weight at most 6, so we hope for such a lucky split; then, there are $\binom{96}{6} \approx 2^{36}$ possible values to enumerate for each side of the above equation, leaving an attack with 2^{37} workfactor and 0.66 probability of success. The attack can be repeated with a different split of e if it is not successful the first time.

6 Related Work

Several authors have analyzed the security of selected randomized countermeasures against side channel attacks [13, 14, 16], including the Oswald-Aigner constructions. However, the analysis techniques previously used have been ad-hoc in sense that they are tailored specifically to the countermeasure being analyzed, and it is not clear how to generalize them to analyze other randomized countermeasures (if this is even possible). In contrast, our techniques are broadly applicable to randomized countermeasures whose operation can be modeled by a probabilistic finite state machine.

Based on a comprehensive case analysis, Okeya and Sakurai [13] present an attack against OA1 that with high probability recovers a 192 bit key using approximately 292 traces of the side channel. They assume the ability to perfectly distinguish between elliptic curve point additions and doublings and do not consider the case when the side channel is noisy.

C.D. Walter [14] presents an attack against OA2 based on a detailed analysis of its operation. With high probability, his attack recovers a 192 bit key using $O(10)$ traces of the side channel. This attack can be generalized to work against OA1 as well. Then, Walter also discusses how to partition traces into smaller subsections and exhaustively search (independently) for the key corresponding to each subsection. Depending on the key size, it is possible for his second technique to succeed with as few as two traces. Both his attacks assume the ability to perfectly distinguish between elliptic curve point additions and doublings.

Song et al. [17] use Hidden Markov Models to exploit weaknesses of the widely used SSH protocol. By observing the inter-keystroke timings of a user's key presses during an SSH session, the authors are able to recover significant information about the key stroke sequences. They use this technique to speed up exhaustive search for passwords by a factor of 50. Other than the work of Song et al., we are not aware of any previous work that uses HMM's for side channel cryptanalysis.

7 Conclusion

We introduced HMM attacks, a general-purpose cryptanalysis technique for evaluating the security properties of randomized countermeasures whose operation can be modeled by a probabilistic finite state machine. We also introduced Input Driven Hidden Markov Models, an extension of HMM's that model inputs, and we presented efficient approximate inference algorithms for recovering the input to an IDHMM given multiple output traces. Our work improves on existing attacks against randomized countermeasures in two fundamental ways. Firstly, previous attacks against randomized countermeasures typically consist of detailed case analyses which are not clear how to generalize to attacks on larger classes of countermeasures. We present a cryptanalytical framework applicable to a general class of randomized countermeasures. Secondly, previous attacks against randomized countermeasures assume the ability to perfectly distinguish between operations in the side channel. Our techniques are still applicable if the side channel is noisy.

We demonstrate the application of HMM attacks and IDHMM's in an analysis of Randomized Addition-Subtraction Chains proposed by Oswald and Aigner. When our

observations of the side channel are perfect, we are able to completely recover the secret key using as few as 5–10 traces. Our attacks are robust to noise in the side channel as well. For instance, when the probability of each observation being incorrect is 0.25, we are still able to recover the secret key by using 50–500 traces.

Acknowledgements

This research was supported in part by NSF ITR CCR-0113941 and NSF CAREER CCR-0093337.

References

1. Ha, J., Moon, S.: Randomized signed-scalar multiplication of ECC to resist power attacks. In: Fourth International Workshop on Cryptographic Hardware and Embedded Systems (CHES). (2002)
2. Oswald, E., Aigner, M.: Randomized addition-subtraction chains as a countermeasure against power attacks. In: Third International Workshop on Cryptographic Hardware and Embedded Systems (CHES). (2001)
3. Izu, T., Takagi, T.: A fast parallel elliptic curve multiplication resistant against side channel attacks. In: PKC2002. (2002)
4. Walter, C.: MIST: An efficient, randomized exponentiation algorithm for resisting power analysis. In: RSA 2002 - Cryptographers' Track. (2002)
5. Liardet, P.Y., Smart, N.: Preventing SPA/DPA in ECC systems using the Jacobi form. In: Third International Workshop on Cryptographic Hardware and Embedded Systems (CHES). (2001)
6. Itoh, K., Yajima, J., Takenaka, M., Torri, N.: DPA countermeasures by improving the window method. In: Fourth International Workshop on Cryptographic Hardware and Embedded Systems (CHES). (2002)
7. May, D., Muller, H., Smart, N.: Randomized register renaming to foil DPA. In: Third International Workshop on Cryptographic Hardware and Embedded Systems (CHES). (2001)
8. Goubin, L., Patarin, J.: DES and differential cryptanalysis. In: First International Workshop on Cryptographic Hardware and Embedded Systems (CHES). (1999)
9. Kocher, P., Jaffe, J., Jun, B.: Differential power analysis. Lecture Notes in Computer Science **1666** (1999) 388–397
10. Kocher, P.C.: Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. Lecture Notes in Computer Science **1109** (1996) 104–113
11. Jordan, M.: An Introduction to Probabilistic Graphical Models. in preparation (2003)
12. Russell, S., Norvig, P.: Artificial Intelligence, A Modern Approach. Prentice Hall (1995)
13. Okeya, K., Sakurai, K.: On insecurity of the side channel attack countermeasure using addition-subtraction chains under distinguishability between addition and doubling. In: The 7th Australasian Conference on Information Security and Privacy. (2002)
14. Walter, C.: Security constraints on the Oswald-Aigner exponentiation algorithm. Cryptology ePrint Archive: http://eprint.iacr.org/ (2003)
15. Coron, J.S.: Resistance against differential power analysis attacks for elliptic curve cryptosystems. In: First International Workshop on Cryptographic Hardware and Embedded Systems (CHES). (1999)
16. Walter, C.: Breaking the Liardet-Smart randomized exponentiation algorithm. In: Fifth Smart Card Research and Advanced Application Conference (CARDIS). (2002)
17. Song, D.X., Wagner, D., Tian, X.: Timing analysis of keystrokes and timing attacks on SSH. In: Tenth USENIX Security Symposium. (2001)

A An Efficient Exact Inference Algorithm for a Single Execution of an IDHMM

In this section we present an efficient dynamic programming algorithm with running time $O(|S|^2 \cdot N)$ for calculating the posterior distributions $\mathbf{Pr}[K_n|y]$ given a single trace of an IDHMM. For the sake of brevity in the derivation, we use $p(x)$ to represent $\mathbf{Pr}[X = x]$ and $p(x|y)$ to represent $\mathbf{Pr}[X = x|Y = y]$.

The goal is to efficiently compute $p(k_n|y)$ for each n. We first use Bayes rule to break apart $p(k_n|y)$:

$$\begin{aligned}
p(k_n|y) &= \frac{p(y|k_n)p(k_n)}{p(y)} \\
&= \frac{\sum_{q_n} p(y, q_n|k_n)p(k_n)}{p(y)} \\
&= \frac{\sum_{q_n} p(y|q_n, k_n)p(q_n, k_n)}{p(y)} \\
&= \frac{\sum_{q_n} p(y_1, y_2, ..., y_n|q_n, k_n)p(y_{n+1}, y_{n+2}, ..., y_N|q_n, k_n)p(q_n, k_n)}{p(y)} \\
&= \frac{\sum_{q_n} p(y_1, y_2, ..., y_n, q_n, k_n)p(y_{n+1}, y_{n+2}, ..., y_N|q_n)}{p(y)} \\
&= \frac{\sum_{q_n} \alpha(q_n, k_n)\beta(q_n)}{p(y)}
\end{aligned}$$

where

$$\begin{aligned}
\alpha(q_n, k_n) &\stackrel{\triangle}{=} p(y_1, y_2, ..., y_n, q_n, k_n) \\
\beta(q_n) &\stackrel{\triangle}{=} p(y_{n+1}, y_{n+2}, ..., y_N|q_n)
\end{aligned}$$

The recursive derivation for the $\alpha(q_n, k_n)$ terms is as follows:

$$\begin{aligned}
\alpha(q_n, k_n) &\stackrel{\triangle}{=} p(y_1, y_2, ..., y_n, q_n, k_n) \\
&= p(y_1, y_2, ..., y_n | q_n, k_n) p(q_n, k_n) \\
&= p(y_n | q_n) p(y_1, y_2, ..., y_{n-1} | q_n, k_n) p(q_n, k_n) \\
&= p(y_n | q_n) p(y_1, y_2, ..., y_{n-1}, q_n, k_n) \\
&= p(y_n | q_n) \sum_{q_{n-1}} \sum_{k_{n-1}} p(y_1, y_2, ..., y_{n-1}, q_n, q_{n-1}, k_n, k_{n-1}) \\
&= p(y_n | q_n) \sum_{q_{n-1}} \sum_{k_{n-1}} p(y_1, y_2, ..., y_{n-1}, q_n, k_n, k_{n-1} | q_{n-1}) p(q_{n-1}) \\
&= p(y_n | q_n) \sum_{q_{n-1}} \sum_{k_{n-1}} p(y_1, y_2, ..., y_{n-1}, k_{n-1} | q_{n-1}) p(q_n, k_n | q_{n-1}) p(q_{n-1}) \\
&= p(y_n | q_n) \sum_{q_{n-1}} \sum_{k_{n-1}} p(y_1, y_2, ..., y_{n-1}, q_{n-1}, k_{n-1}) p(q_n | q_{n-1}, k_n) p(k_n) \\
&= p(y_n | q_n) \sum_{q_{n-1}} \sum_{k_{n-1}} \alpha(q_{n-1}, k_{n-1}) p(q_n | q_{n-1}, k_n) p(k_n)
\end{aligned}$$

The recursive derivation for the $\beta(q_n)$ terms is as follows:

$$\begin{aligned}
\beta(q_n) &\stackrel{\triangle}{=} p(y_{n+1}, y_{n+2}, ..., y_N | q_n) \\
&= \sum_{q_{n+1}} p(y_{n+1}, y_{n+2}, ..., y_N, q_{n+1} | q_n) \\
&= \sum_{q_{n+1}} p(y_{n+1}, y_{n+2}, ..., y_N | q_{n+1}, q_n) p(q_{n+1} | q_n) \\
&= \sum_{q_{n+1}} p(y_{n+1}, y_{n+2}, ..., y_N | q_{n+1}) p(q_{n+1} | q_n) \\
&= \sum_{q_{n+1}} p(y_{n+1} | q_{n+1}) p(y_{n+2}, ..., y_N | q_{n+1}) \sum_{k_{n+1}} p(q_{n+1}, k_{n+1} | q_n) \\
&= \sum_{q_{n+1}} p(y_{n+1} | q_{n+1}) \beta(q_{n+1}) \sum_{k_{n+1}} p(q_{n+1} | q_n, k_{n+1}) p(k_{n+1})
\end{aligned}$$

By convention, $\beta(q) = 1$ for all $q \in S$ if $n = N$. The α recursion is initialized as follows:

$$\alpha(q_1, k_1) = p(y_1 | q_1) p(q_1 | q_0, k_1) p(k_1)$$

where q_0 is the start state of the IDHMM.