Lecture Note 9

ATTACKS ON CRYPTOSYSTEMS II

Sourav Mukhopadhyay

Cryptography and Network Security - MA61027

Birthday attack

- The Birthday attack makes use of what's known as the Birthday paradox to try to attack cryptographic hash functions.
- The birthday paradox can be stated as follows: What is the minimum value of k such that the probability is greater than 0.5 that at least two people in a group of k people have the same birthday?

- It turns out that the answer is 23 which is quite a surprising result. In other words if there are 23 people in a room, the probability that two of them have the same birthday is approximately 0.5.
- If there is 100 people (i.e. k=100) then the probability is .9999997, i.e. you are almost guaranteed that there will be a duplicate. A graph of the probabilities against the value of k is shown in figure 1.

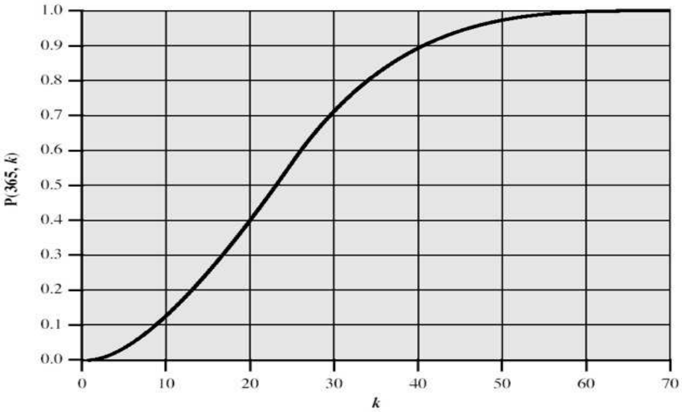


Figure 1: The Birthday Paradox.

- Although this is the case for birthdays we can generalise it for n equally likely values (in the case of birthdays $n = 365$).
- So the problem can be stated like this: Given a random variable that is an integer with uniform distribution between 1 and n and a selection of k instances ($k \leq n$) of the random variable, what is the probability $P(n, k)$, that there is at least one duplicate?

- It turns out that this value is

$$
\begin{aligned}
P(n,k) &= 1 - \frac{n!}{(n-k)!n^k} \\
&= 1 - \left[(1-\frac{1}{n}) \times (1-\frac{2}{n}) \times \ldots \times (1-\frac{k-1}{n})\right]
\end{aligned} \tag{1}
$$

- Take it that the following inequality holds

$$
(1-x) \leq e^{-x} \tag{2}
$$

- Then we have

$$
\begin{aligned}
P(n,k) &> 1-[(e^{-1/n})\times(e^{-2/n})\times\ldots\times(e^{-(k-1)/n})] \\
&> 1-e^{-[(1/n)+(2/n)+((k-1)/n)]} \\
&> 1-e^{-(k\times(k-1))/2n}
\end{aligned}
\tag{3}
$$

- We would like to know when $P(n,k) > 0.5$ so we set the right hand side of equation 3 to 0.5:

$$
\begin{aligned}
0.5 &= 1-e^{-(k\times(k-1))/2n} \\
\Rightarrow ln(2) &= \frac{k\times(k-1)}{2n}
\end{aligned}
\tag{4}
$$

- For large values of k we can replace $k \times (k-1)$ by k^2 giving

$$k = \sqrt{2(ln2)n} = 1.18\sqrt{n} \approx \sqrt{n} \tag{5}$$

- Which can be seen to be almost equal to 23 for $n = 365$.

- Now lets look at this in terms of hash codes.
- Remember a hash code is a function that takes a variable length message M and produces a fixed length message digest.
- Assuming the length of the digest is m then there are 2^m possible message digests.
- Normally however, because the length of M will generally be greater than m this implies that more than one message will be mapped to the same digest.
- Of course the idea is to make it computationally infeasible to find two messages that map to the same digest.

- However if we apply k random messages to our hash code what must the value of k be so that there is the probability of 0.5 that at least one duplicate (i.e. $H(x) = H(y)$ will occur for some inputs x, y)?
- This is the same as the question we asked about the birthday duplicates.
- Using equation 5 we have

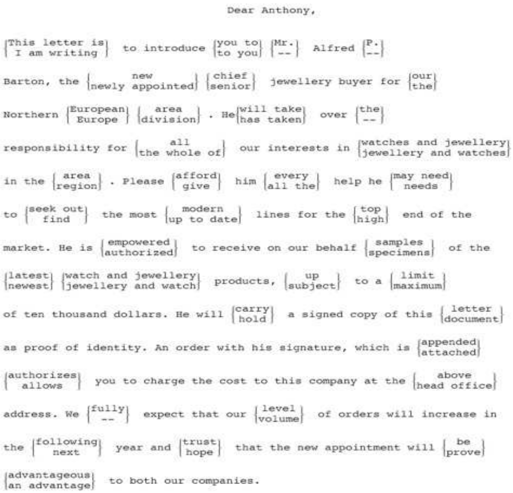
$$k = \sqrt{2^m} = 2^{m/2} \tag{6}$$

- Using this idea we can discuss the birthday attack as follows:
  - The source, A is prepared to “sign” a message by appending the appropriate m-bit hash code and encrypting that hash code with A’s private key.
  - The opponent generates $2^{m/2}$ variations on the message, all of which convey essentially the same meaning. The opponent prepares an equal number of messages, all of which are variations of the fraudulent message to be substituted for the real one.

- The two sets of messages are compared to find a pair of messages that produce the same hash code. The probability of success is greater than 0.5. If no match is found, additional valid and fraudulent messages are generated until a match is made.
- The opponent offers the valid variation to A for signature. This signature can then be attached to the fraudulent variation for transmission to the intended recipient. Because the two variations have the same hash code, they will produce the same signature; the opponent is assured of success even though the

encryption key is not known.

- If we use a 64-bit hash code then the level of effort required is only on the order of $2^{m/2} = 2^{64/2} = 2^{32}$ which is clearly not sufficient to withstand today's computational systems.
- The generation of many variations that convey the same meaning is not that difficult as figure 2 shows.



Dear Anthony,

{This letter is | I am writing} to introduce {you to | to you} {Mr. | --} Alfred {P. | --} Barton, the {new | newly appointed} {chief | senior} jewellery buyer for {our | the} Northern {European | Europe} {area | division}. He {will take | has taken} over {the | --} responsibility for {all | the whole of} our interests in {watches and jewellery | jewellery and watches} in the {area | region}. Please {afford | give} him {every | all the} help he {may need | needs} to {seek out | find} the most {modern | up to date} lines for the {top | high} end of the market. He is {empowered | authorized} to receive on our behalf {samples | specimens} of the {latest | newest} {watch and jewellery | jewellery and watch} products, {up | subject} to a {limit | maximum} of ten thousand dollars. He will {carry | hold} a signed copy of this {letter | document} as proof of identity. An order with his signature, which is {appended | attached} {authorizes | allows} you to charge the cost to this company at the {above | head office} address. We {fully | --} expect that our {level | volume} of orders will increase in the {following | next} year and {trust | hope} that the new appointment will {be | prove} {advantageous | an advantage} to both our companies.

Figure 2: A letter in 2^{37} variations.

Implementation Attacks

- Implementation attacks take on a different approach to the above for discovering the secret key.
- Instead of attacking the mathematical properties of the algorithm these form of attacks (also known as side channel attacks) take advantage of the physical phenomena that occurs when a cryptographic algorithm is implemented in hardware.

- Four side channel attacks are listed in the FIPS standard 140-2 “Security Requirements for Cryptographic Modules”, **Power Analysis**, **Timing Analysis**, **Fault Induction** and **TEMPEST**.
- Here we will be interested mainly in Differential Power Analysis (DPA) as it applies to DES however we will have a brief look at Timing attacks.

Differential Power Analysis

- Power Analysis is a relatively new concept but has proven to be quite effective in attacking smartcards and similar devices.
- The smartcard is very susceptible to this form of attack mainly because it applies little or no power filtering due to its small size.
- It was first demonstrated by Ernst Bovelander in 1997 but a specific attack strategy was not given.

- A year later it was brought to the general public's attention by Paul Kocher and the Cryptographic Research team in San Francisco.
- Kocher et al. provided an attack strategy that would recover the secret key from cryptographic systems running the DES algorithm.
- This caused great concern amongst the smartcard community and a search for an effective countermeasure began.

- To date a limited number of countermeasures have been proposed and none are fully effective.
- The attacks work equally well on other cryptographic algorithms as shown by Thomas Messerges et al. who presented a great deal of supplementary research on the subject.
- Power analysis involves an analysis of the pattern of power consumed by a cryptographic module as it performs its operations.

- The purpose of this pattern analysis is to acquire knowledge about causal operations that is not readily available through other sources.
- The power consumption will generally be different for each operation performed (and even for the same operations with different data values).
- One of the causes of these variations is the transistor technology used to implement the module.
- The transistors act as voltage controlled switches, and the power they consume varies with the type of instructions being processed.

- For example, a conditional branch instruction appears to cause a lot of noticeable fluctuations according to Kocher, and should therefore be avoided if possible where secret keys are concerned.
- An example of a setup for a power analysis attack is shown in figure 3.
- For smartcards and similar devices, the power can be measured across a $10 - 50\Omega$ resistor in series with the power or ground line of the specific device.

- The resistor should be small enough so as not to interfere with the operation of the circuit itself, but large enough to give easily observable voltage fluctuations. It is better to put the resistor in series with the ground of the device.
- If the power line is used then two scope probes would be needed and the resultant waveforms substracted.

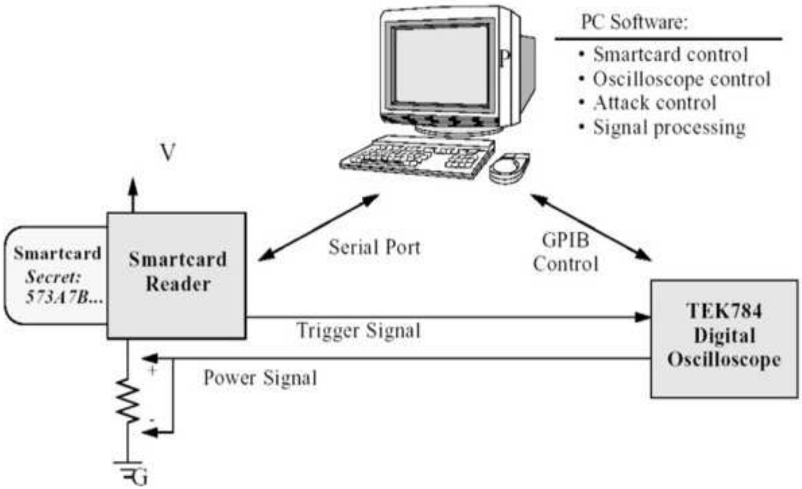


Figure 3: An example setup for a Differential Power Analysis attack on a smartcard.

- Although the setup in figure 3 will suffice for a smartcard it will generally not be this simple for a complex cryptographic accelerator which probably draws its power from the peripheral component interconnect (PCI) backplane of a computer.
- Ideally, the attacker would wish to get as close as possible to the actual chip performing the operations if a high signal to noise ratio (SNR) is to be obtained.

- This might be more difficult than it first appears as information on which of the boards numerous chips is actually running the algorithm may not be readily available.
- Even if it were, the power pin of the chip would have to be physically separated from the board to perform the attack and then reattached once complete (if the attack were to go unnoticed).
- Most tamper resistant devices would not permit this from happening.
- An example of a possible setup is shown in figure 4.

- In this case a PCI extender board is used to measure the power fluctuations.
- The actual cryptographic board slots into the extender board and therefore the power the cryptographic board draws from the PCI backplane has to flow through the extender board which can be fitted with some points that allow for measurement of the power.
- These can be home made or easily purchased.

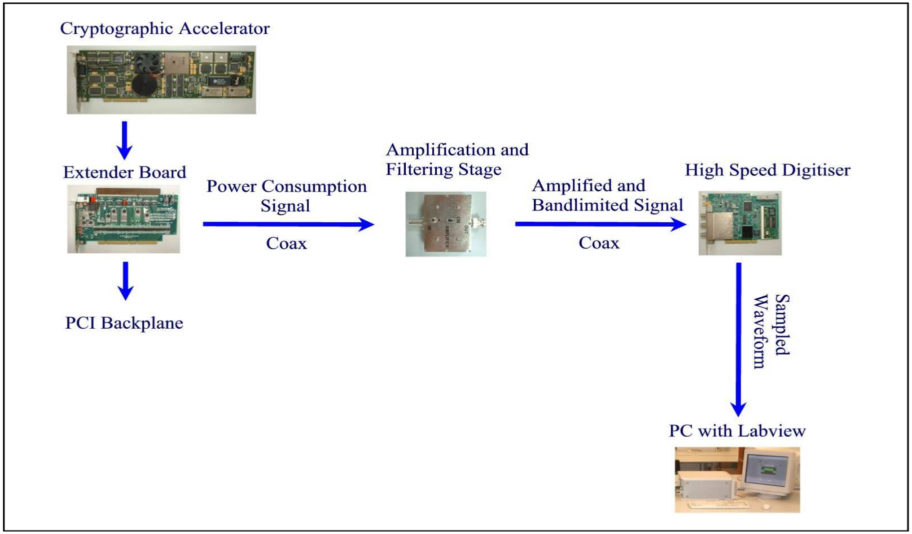


Figure 4: An example setup for a Differential Power Analysis attack on a high speed cryptographic accelerator.

- Assuming a setup such as those in figures 3 and 4 in which the algorithm being executed is the Data Encryption Standard (DES) the attack can proceed as follows.
- A method must be devised to produce a random set of J plaintext inputs that can be sent to the cryptosystem for encryption.
- This method must be automated as the number of random plaintext inputs will be quite large. Generally this will be the job of the PC however on more complex cryptosystems it may be possible to upload new firmware that will do the trick.

- On receiving these plaintext inputs, pi_j, $1 \leq j \leq J$, the board will begin to run its algorithm and draw varying amounts of power.
- These power fluctuations can be sampled using a digital sampling oscilloscope which should be capable of sampling at about 20-30 times the clock frequency being used.
- There are two main reasons for this:
  1. Possible that we might have multiple operations occuring in each clock cycle. Also, operation of interest might only last a small fraction of the clock

cycle.

2. The more samples you have per cycle the less chance of noise caused by a misalignment of samples.

- The waveforms observed for each pi_j can be represented as a matrix wf_{jk}, where $1 \leq k \leq K$.
- The subscripts j and k are used to identify the plaintext number causing the waveform and the time sample point within that particular waveform, respectively.
- A second column matrix, co_j, can also be used to represent the ciphertext output.

- In practice, each row of wf_{jk} would probably be stored as a separate file for ease of processing.
- Having captured each power waveform and ciphertext output, a function known as a **partitioning function**, D(.), must now be defined.
- This function will allow division of the matrix wf_{jk} into two sub-matrices $wf0_{pk}$ and $wf1_{qk}$ containing P and Q rows respectively, with $1 \leq p \leq P$ and $1 \leq q \leq Q$ where $P + Q = J$.
- Provided that the inputs pi_j were randomly produced,

then $P = Q = J/2$ as $J \rightarrow \infty$ (i.e. the waveforms will be divided equally between the two sets).

- The partitioning function allows the division of wf_{jk} because it calculates the value of a particular bit, at particular times, during the operation of the algorithm.
- If the value of this bit is known, then it will also be known whether or not a power bias should have occurred in the captured waveform.
- For a 1, a bias should occur, and for a 0 it shouldn't.
- Separating the waveforms into two separate matrices (one

in which the bias occurred and another in which it didn't) will allow averaging to reduce the noise and enhance the bias (if it occurred).

- For randomly chosen plaintexts, the output of the D(.) function will equal either a 1 or 0 with probability $\frac{1}{2}$ (this is just another statement of the fact that $P = Q = J/2$ as $J \to \infty$).
- An example of a partitioning function is:

$$D(C_1, C_6, K_{16}) = C_1 \oplus SBOX1(C_6 \oplus K_{16}) \tag{7}$$

- Where SBOX1(.) is a function that outputs the target bit of S-box 1 in the last round of DES (in this case it's the first bit), C_1 is the one bit of co_j that is exclusive OR'ed with this bit, C_6 is the 6 bits of co_j that is exclusive OR'ed with the last rounds subkey and K_{16} is the 6 bits of the last round's subkey that is input into S-box 1.

- The value of this partitioning function must be calculated at some point throughout the algorithm.
- So, if the values C_1, C_6 and K_{16} can be determined, it will be known whether or not a power bias occurred in each waveform.

- It is assumed that the values C_1 and C_6 can be determined and the value of the subkey K_{16} is the information sought.
- To find this, an exhaustive search needs to be carried out. As it is 6 bits long, a total of $2^6 = 64$ subkeys will need to be tested.
- The right one will produce the correct value of the partitioning bit for every plaintext input.
- However, the incorrect one will only produce the correct result with probability $\frac{1}{2}$.

- In this case, the two sets $wf0_{pk}$ and $wf1_{qk}$ will contain a randomly distributed collection of waveforms which will average out to the same result (Provided the plaintext inputs are randomly chosen).
- The differential trace (discussed below) will thus show a power bias for the correct key only.
- Of course it means that 64 differential traces are needed but this is a vast improvement over a brute force search of the entire 56 bit key.
- Mathematically, the partitioning of wf_{jk} can be

represented as

$$wf0_{pk} = \{wf_{jk}|D(.) = 0\} \tag{8}$$

and

$$wf1_{qk} = \{wf_{jk}|D(.) = 1\} \tag{9}$$

- Once the matrices $wf0_{pk}$ and $wf1_{qk}$ have been set up, the average of each is then taken producing two waveforms $awf0_k$ and $awf1_k$ both consisting of K samples.
- By taking the averages of each, the noise gets reduced to very small levels but the power spikes in $wf1_{pk}$ will be reinforced.

- However, averaging will not reduce any periodic noise contained within the power waveforms and inherent to the operations on the cryptographic board.
- This can largely be eliminated by subtracting $awf0_{pk}$ from $awf1_{qk}$ (this can be thought of as demodulating a modulated signal to reveal the "baseband", where the periodic noise is the "carrier").
- The only waveform remaining will be the one with a number of bias points identifying the positions where the target bit was manipulated.

- This trace is known as a **differential trace**, ΔD_k.
- Again, in mathematical terms, the above can be stated as

$$awf0_k = \frac{1}{P} \sum_{wf_{jk} \in wf1} wf_{jk} = \frac{1}{P} \sum_{p=1}^{P} wf0_{pk} \qquad (10)$$

and

$$awf1_k = \frac{1}{Q} \sum_{wf_{jk} \in wf0} wf_{jk} = \frac{1}{Q} \sum_{q=1}^{Q} wf1_{qk} \qquad (11)$$

- The differential trace ΔD_k is then obtained as

$$\Delta D_k = awf1_k - awf0_k \qquad (12)$$

- The last five equations can now be condensed into one:

$$\Delta D_k = \frac{\Sigma_{j=1}^{J} D(.)wf_{jk}}{\Sigma_{j=1}^{J} D(.)} - \frac{\Sigma_{j=1}^{J}(1 - D(.))wf_{jk}}{\Sigma_{j=1}^{J}(1 - D(.))} \quad (13)$$

- As $J \longrightarrow \infty$, the power biases will average out to a value ϵ which will occur at times k_D - each time the target bit D was manipulated.

- In this limit, the averages $awf0_k$ and $awf1_k$ will tend toward the expectation $E\{wf0_k\}$ and $E\{wf1_k\}$, and equations 12 and 13 will converge to

$$E\{wf1_k\} - E\{wf0_k\} = \epsilon, \qquad \text{at times} \quad k = k_D \quad (14)$$

and

$$E\{wf1_k\} - E\{wf0_k\} = 0, \qquad \text{at times} \quad k \neq k_D \quad (15)$$

- Therefore, at times $k = k_D$, there will be a power bias ϵ visible in the differential trace. At all other times, the power will be independent of the target bit and the differential trace will tend towards 0.
- The above will only work if the subkey guess was correct. For all other guesses the partitioning function will separate the waveforms randomly, and equations 14 and

15 will condense to

$$E\{wf1_k\} - E\{wf0_k\} = 0, \qquad \forall k \tag{16}$$

- As mentioned above, 64 differential traces are needed to determine which key is the correct one.
- Theoretically, the one containing bias spikes will allow determination of the correct key however, in reality the other waveforms will contain small spikes due to factors such as non-random choices of plaintext inputs, statistical biases in the S-boxes and a non-infinite number of waveforms collected.

- Generally however, the correct key will show the largest bias spikes and can still be determined quite easily.
- The other 42 bits from the last round's subkey can be determined by applying the same method to the other 7 S-boxes.
- A brute force search can then be used to obtain the remaining 8 bits of the 56 bit key.
- NOTE: The same J power signals can be used for each S-box as the different D functions re-order them accordingly.

Mitigation Techniques

- The following could be used as mitigation techniques for power attacks in general:

  1. **Timing Randomisation:** This involves placing random time delays into the software so that a power analysis will not be possible. With random delays introduced, a steady trigger will not be sufficient to allow the averaging to work and will therefore act as a countermeasure.

2. **Internal power supplies/power supply filtering:** This would be another method that could be used to reduce the possibility of a power attack. For example, Adi Shamir proposes building a simple capacitance network into each smartcard to allow the fluctuations to be contained within the smartcard itself thereby preventing power attacks.
3. **Data masking:** One of the methods proposed consists of *masking* the intermediate data (i.e. mask the input data and key before executing the algorithm). This would make the power fluctuations

independent of the actual data.

4. **Tamper Resistance:** This involves placing some detection/prevention system around the device to stop intruders gaining access to the power fluctuations.
5. **Fail Counters:** A differential power analysis attack requires the attacker to obtain a significant number of power waveforms. In order to do this the attacker must have the ability to run quite a few encryptions on the system under attack. If the number of encryptions were limited to a certain number then the

attacks would become increasingly difficult.

6. **Removal of conditional elements:** One of the main features used to attack the square and multiply algorithm is the fact that it has a conditional multiplication that depends on the value of the exponent bit being operated upon. One suggested countermeasure is to implement this multiplication in every round (regardless of the value of the bit) and to only do a register update when the bit is a 1.

Timing Attacks

- A timing attack is somewhat analogous to a burglar guessing the combination of a safe by observing how long it takes for someone to turn the dial from number to number.
- We can explain the attack using the modular exponentiation algorithm shown in figure 5, but the attack can be adapted to work with any implementation that does not run in fixed time.

- In this algorithm, modular exponentiation is accomplished bit by bit, with one modular multiplication performed at each iteration and an additional modular multiplication performed for each 1 bit.

```
square_and_mul(b, e, m)
{
    d = 1;
    for (k = N-1 downto 0)
    {
        d = (d × d) mod m;
        if (e[k] == 1)
        {
            d = (d × b) mod m;
        }
    Return d;
}
```

Figure 5: Square and Multiply algorithm for Computing b^e mod (m) where e is N bits long.

- As Kocher points out in his paper, the attack is simplest to understand in an extreme case.
- Suppose the target system uses a modular multiplication function that is very fast in almost all cases but in a few cases takes musch more time than an entire average modular exponentiation.
- The attack proceeds bit by bit starting with the leftmost bit $e[N-1]$. Suppose that the first j bits are known (to obtain the entire exponent, start with $j = 0$ and repeat the attack until the entire exponent is known).

- For a given ciphertext, the attacker can complete the first j iterations of the **for** loop.
- Operation of subsequent steps depends on the unkown exponent bit.
- If the bit is set d = (d $\times$ b) mod m will be executed.
- For a few values of b and d, the modular multiplication will be extremely slow, and the attacker knows which these are.
- Therefore, if the observed time to execute the decryption algorithm is always slow when this particular iteration is

slow with a 1 bit, then this bit is assumed to be 1.

- If a number of observed execution times for the entire algorithm are fast, then this bit is assumed to be 0.
- In practice, modular exponentiation implementations do not have such extreme timing variations, in which the execution time of a single iteration can exceed the mean execution time of the entire algorithm.
- Nevertheless, there is enough variation to make this attack practical.
- Although the timing attack is a serious threat, there are

simple counter measures that can be used including the following:

– **Constant exponentiation time:** Ensure that all exponentiations take the same amount of time before returning a result. This is a simple fix but does degrade performance.

– **Random delay:** Better performance could be achieved by adding a random delay to the exponentiation algorithm to confuse the timing attack. Kocher point out that if defenders don't add enough noise, attackers could still succeed by collecting additional measurements to compensate for the random delays.

- **Blinding:** Multiply the ciphertext by a random number before performing exponentiation. This process prevents the attacker from knowing what ciphertext bits are being processed inside the computer and therefore prevents the bit-by-bit analsys essential to the timing attack.

• RSA Data Security incorportates a blinding feature into some of its products. The private-key operation $M = C^d \bmod n$ is implemented as follows:

1. Generate a secret random number r between 0 and $n - 1$.
2. Compute $C' = C(r^e) \bmod n$, where e is the public exponent.
3. Comput $M' = (C')^d \bmod n$ with the ordinary RSA implementation.
4. Compute $M = M'r^{-1} \bmod n$ (where r^{-1} is the multiplicative inverse of $r \bmod n$). It can be

demonstrated that this is the correct result by observing that $r^{ed} \bmod n = r \bmod n$.

- RSA Data Security reports a 2 to 10% performance penalty for blinding.