3.6 SECRET SPLITTING

Imagine that you've invented a new, extra gooey, extra sweet, cream filling or a burger sauce that is even more tasteless than your competitors'. This is important; you have to keep it secret. You could tell only your most trusted employees the exact mixture of ingredients, but what if one of them defects to the competition? There goes the secret, and before long every grease palace on the block will be making burgers with sauce as tasteless as yours.

This calls for **secret splitting**. There are ways to take a message and divide it up into pieces [551]. Each piece by itself means nothing, but put them together and the message appears. If the message is the recipe and each employee has a piece, then only together can they make the sauce. If any employee resigns with his single piece of the recipe, his information is useless by itself.

The simplest sharing scheme splits a message between two people. Here's a protocol in which Trent can split a message between Alice and Bob:

(1) Trent generates a random-bit string, R, the same length as the message, M.

(2) Trent XORs M with R to generate S.

$$M \oplus R = S$$

(3) Trent gives R to Alice and S to Bob.

To reconstruct the message, Alice and Bob have only one step to do:

(4) Alice and Bob XOR their pieces together to reconstruct the message:

$$R \oplus S = M$$

This technique, if done properly, is absolutely secure. Each piece, by itself, is absolutely worthless. Essentially, Trent is encrypting the message with a one-time pad and giving the ciphertext to one person and the pad to the other person. Section 1.5 discusses one-time pads; they have perfect security. No amount of computing power can determine the message from one of the pieces.

It is easy to extend this scheme to more people. To split a message among more than two people, XOR more random-bit strings into the mixture. In this example, Trent divides up a message into four pieces:

(1) Trent generates three random-bit strings, R, S, and T, the same length as the message, M.

(2) Trent XORs M with the three strings to generate U:

$$M \oplus R \oplus S \oplus T = U$$

(3) Trent gives R to Alice, S to Bob, T to Carol, and U to Dave.

Alice, Bob, Carol, and Dave, working together, can reconstruct the message:

(4) Alice, Bob, Carol, and Dave get together and compute:

$$R \oplus S \oplus T \oplus U = M$$

This is an adjudicated protocol. Trent has absolute power and can do whatever he wants. He can hand out gibberish and claim that it is a valid piece of the secret; no one will know it until they try to reconstruct the secret. He can hand out a piece to Alice, Bob, Carol, and Dave, and later tell everyone that only Alice, Carol, and Dave are needed to reconstruct the secret, and then fire Bob. But since this is Trent's secret to divide up, this isn't a problem.

However, this protocol has a problem: If any of the pieces gets lost and Trent isn't around, so does the message. If Carol, who has a piece of the sauce recipe, goes to work for the competition and takes her piece with her, the rest of them are out of luck. She can't reproduce the recipe, but neither can Alice, Bob, and Dave working together. Her piece is as critical to the message as every other piece combined. All Alice, Bob, or Dave know is the length of the message—nothing more. This is true because R, S, T, U, and M all have the same length; seeing anyone of them gives the length of M. Remember, M isn't being split in the normal sense of the word; it is being XORed with random values.

3.7 Secret Sharing

You're setting up a launch program for a nuclear missile. You want to make sure that no single raving lunatic can initiate a launch. You want to make sure that no two raving lunatics can initiate a launch. You want at least three out of five officers to be raving lunatics before you allow a launch.

This is easy to solve. Make a mechanical launch controller. Give each of the five officers a key and require that at least three officers stick their keys in the proper slots before you'll allow them to blow up whomever we're blowing up this week. (If you're really worried, make the slots far apart and require the officers to insert the keys simultaneously—you wouldn't want an officer who steals two keys to be able to vaporize Toledo.)

We can get even more complicated. Maybe the general and two colonels are authorized to launch the missile, but if the general is busy playing golf then five colonels are required to initiate a launch. Make the launch controller so that it requires five keys. Give the general three keys and the colonels one each. The general together with any two colonels can launch the missile; so can the five colonels. However, a general and one colonel cannot; neither can four colonels.

A more complicated sharing scheme, called a **threshold scheme**, can do all of this and more—mathematically. At its simplest level, you can take any message (a secret recipe, launch codes, your laundry list, etc.) and divide it into n pieces, called **shadows** or shares, such that any m of them can be used to reconstruct the message. More precisely, this is called an **(m,n)-threshold scheme**.

With a (3,4)-threshold scheme, Trent can divide his secret sauce recipe among Alice, Bob, Carol, and Dave, such that any three of them can put their shadows together and reconstruct the message. If Carol is on vacation, Alice, Bob, and Dave can do it. If Bob gets run over by a bus, Alice, Carol, and Dave can do it. However, if Bob gets run over by a bus while Carol is on vacation, Alice and Dave can't reconstruct the message by themselves.

Alice isn't keen on letting Bob know her x. How can Alice let Bob compute $f(x)$ for her without telling him x?

This is the general problem of **computing with encrypted data**, also called **hiding information from an oracle**. (Bob is the oracle; he answers questions.) There are ways to do this for certain functions; they are discussed in Section 23.6.

4.9 BIT COMMITMENT

The Amazing Alice, magician extraordinaire, will now perform a mystifying feat of mental prowess. She will guess the card Bob will choose before he chooses it! Watch as Alice writes her prediction on a piece of paper. Marvel as Alice puts that piece of paper in an envelope and seals it shut. Thrill as Alice hands that sealed envelope to a random member of the audience. "Pick a card, Bob, any card." He looks at it and shows it to Alice and the audience. It's the seven of diamonds. Alice now takes the envelope back from the audience. She rips it open. The prediction, written before Bob chose his card, says "seven of diamonds"! Applause.

To make this work, Alice had to switch envelopes at the end of the trick. However, cryptographic protocols can provide a method immune from any sleight of hand. Why is this useful? Here's a more mundane story:

Stockbroker Alice wants to convince investor Bob that her method of picking winning stocks is sound.

BOB: "Pick five stocks for me. If they are all winners, I'll give you my business."

ALICE: "If I pick five stocks for you, you could invest in them without paying me. Why don't I show you the stocks I picked last month?"

BOB: "How do I know you didn't change last month's picks after you knew their outcome? If you tell me your picks now, I'll know that you can't change them. I won't invest in those stocks until after I've purchased your method. Trust me."

ALICE: "I'd rather show you my picks from last month. I didn't change them. Trust me."

Alice wants to commit to a prediction (i.e., a bit or series of bits) but does not want to reveal her prediction until sometime later. Bob, on the other hand, wants to make sure that Alice cannot change her mind after she has committed to her prediction.

Bit Commitment Using Symmetric Cryptography

This bit-commitment protocol uses symmetric cryptography:

(1) Bob generates a random-bit string, R, and sends it to Alice.

R

(2) Alice creates a message consisting of the bit she wishes to commit to, b (it can actually be several bits), and Bob's random string. She encrypts it with some random key, K, and sends the result back to Bob.

$E_K(R,b)$

That is the commitment portion of the protocol. Bob cannot decrypt the message, so he does not know what the bit is.

When it comes time for Alice to reveal her bit, the protocol continues:

(3) Alice sends Bob the key.

(4) Bob decrypts the message to reveal the bit. He checks his random string to verify the bit's validity.

If the message did not contain Bob's random string, Alice could secretly decrypt the message she handed Bob with a variety of keys until she found one that gave her a bit other than the one she committed to. Since the bit has only two possible values, she is certain to find one after only a few tries. Bob's random string prevents her from using this attack; she has to find a new message that not only has her bit inverted, but also has Bob's random string exactly reproduced. If the encryption algorithm is good, the chance of her finding this is minuscule. Alice cannot change her bit after she commits to it.

Bit Commitment Using One-Way Functions

This protocol uses one-way functions:

(1) Alice generates two random-bit strings, R_1 and R_2.

R_1, R_2

(2) Alice creates a message consisting of her random strings and the bit she wishes to commit to (it can actually be several bits).

(R_1, R_2, b)

(3) Alice computes the one-way function on the message and sends the result, as well as one of the random strings, to Bob.

$H(R_1, R_2, b), R_1$

This transmission from Alice is evidence of commitment. Alice's one-way function in step (3) prevents Bob from inverting the function and determining the bit.

When it comes time for Alice to reveal her bit, the protocol continues:

(4) Alice sends Bob the original message.

(R_1, R_2, b)

(5) Bob computes the one-way function on the message and compares it and R_1, with the value and random string he received in step (3). If they match, the bit is valid.

The benefit of this protocol over the previous one is that Bob does not have to send any messages. Alice sends Bob one message to commit to a bit and another message to reveal the bit.

Bob's random string isn't required because the result of Alice's commitment is a message operated on by a one-way function. Alice cannot cheat and find another

message (R_1,R_2',b'), such that $H(R_1,R_2',b') = H(R_1,R_2,b)$. By sending Bob R_1 she is committing to the value of b. If Alice didn't keep R_2 secret, then Bob could compute both $H(R_1,R_2,b)$ and $H(R_1,R_2,b')$ and see which was equal to what he received from Alice.

Bit Commitment Using Pseudo-Random-Sequence Generators

This protocol is even easier [1137]:

(1) Bob generates a random-bit string and sends it to Alice.

R_B

(2) Alice generates a random seed for a pseudo-random-bit generator. Then, for every bit in Bob's random-bit string, she sends Bob either:

(a) the output of the generator if Bob's bit is 0, or

(b) the XOR of output of the generator and her bit, if Bob's bit is 1.

When it comes time for Alice to reveal her bit, the protocol continues:

(3) Alice sends Bob her random seed.

(4) Bob completes step (2) to confirm that Alice was acting fairly.

If Bob's random-bit string is long enough, and the pseudo-random-bit generator is unpredictable, then there is no practical way Alice can cheat.

Blobs

These strings that Alice sends to Bob to commit to a bit are sometimes called **blobs**. A blob is a sequence of bits, although there is no reason in the protocols why it has to be. As Gilles Brassard said, "They could be made out of fairy dust if this were useful" [236]. Blobs have these four properties:

1. Alice can commit to blobs. By committing to a blob, she is committing to a bit.
2. Alice can open any blob she has committed to. When she opens a blob, she can convince Bob of the value of the bit she committed to when she committed to the blob. Thus, she cannot choose to open any blob as either a zero or a one.
3. Bob cannot learn how Alice is able to open any unopened blob she has committed to. This is true even after Alice has opened other blobs.
4. Blobs do not carry any information other than the bit Alice committed to. The blobs themselves, as well as the process by which Alice commits to and opens them, are uncorrelated to anything else that Alice might wish to keep secret from Bob.

5.3 BLIND SIGNATURES

An essential feature of digital signature protocols is that the signer knows what he is signing. This is a good idea, except when we want the reverse.

We might want people to sign documents without ever seeing their contents. There are ways that a signer can *almost*, but not exactly, know what he is signing. But first things first.

Completely Blind Signatures

Bob is a notary public. Alice wants him to sign a document, but does not want him to have any idea what he is signing. Bob doesn't care what the document says; he is just certifying that he notarized it at a certain time. He is willing to go along with this.

(1) Alice takes the document and multiplies it by a random value. This random value is called a **blinding factor**.

(2) Alice sends the blinded document to Bob.

(3) Bob signs the blinded document.

(4) Alice divides out the blinding factor, leaving the original document signed by Bob.

This protocol only works if the signature function and multiplication are commutative. If they are not, there are other ways to modify the document other than by multiplying. Some relevant algorithms appear in Section 23.12. For now, assume that the operation is multiplication and all the math works.

Can Bob cheat? Can he collect any information about the document that he is signing? If the blinding factor is truly random and makes the blinded document truly random, he cannot. The blinded document Bob signs in step (2) looks nothing like the document Alice began with. The blinded document with Bob's signature on it in step (3) looks nothing like the signed document at the end of step (4). Even if Bob got his hands on the document, with his signature, after completing the protocol, he cannot prove (to himself or to anyone else) that he signed it in that particular protocol. He knows that his signature is valid. He can, like anyone else, verify his signature. However, there is no way for him to correlate any information he received during the signing protocol with the signed document. If he signed a million documents using this protocol, he would have no way of knowing in which instance he signed which document.

The properties of completely blind signatures are:

1. Bob's signature on the document is valid. The signature is a proof that Bob signed the document. It will convince Bob that he signed the document if it is ever shown to him. It also has all of the other properties of digital signatures discussed in Section 2.6.
2. Bob cannot correlate the signed document with the act of signing the document. Even if he keeps records of every blind signature he makes, he cannot determine when he signed any given document.

Eve, who is in the middle, watching this protocol, has even less information than Bob.

Blind Signatures

With the completely blind signature protocol, Alice can have Bob sign anything: "Bob owes Alice a million dollars," "Bob owes Alice his first-born child," "Bob owes Alice a bag of chocolates." The possibilities are endless. This protocol isn't useful in many applications.

However, there is a way that Bob can know what he is signing, while still maintaining the useful properties of a blind signature. The heart of this protocol is the cut-and-choose technique. Consider this example. Many people enter this country every day, and the Department of Immigration wants to make sure they are not smuggling cocaine. The officials could search everyone, but instead they use a probabilistic solution. They will search one-tenth of the people coming in. One person in ten has his belongings inspected; the other nine get through untouched. Chronic smugglers will get away with their misdeeds most of the time, but they have a 10 percent chance of getting caught. And if the court system is effective, the penalty for getting caught once will more than wipe out the gains from the other nine times.

If the Department of Immigration wants to increase the odds of catching smugglers, they have to search more people. If they want to decrease the odds, they have to search fewer people. By manipulating the probabilities, they control how successful the protocol is in catching smugglers.

The blind signature protocol works in a similar manner. Bob will be given a large pile of different blinded documents. He will **open**, that is examine, all but one and then sign the last.

Think of the blinded document as being in an envelope. The process of blinding the document is putting the document in an envelope and the process of removing the blinding factor is opening the envelope. When the document is in an envelope, nobody can read it. The document is signed by having a piece of carbon paper in the envelope: When the signer signs the envelope, his signature goes through the carbon paper and signs the document as well.

This scenario involves a group of counterintelligence agents. Their identities are secret; not even the counterintelligence agency knows who they are. The agency's director wants to give each agent a signed document stating: "The bearer of this signed document, (insert agent's cover name here), has full diplomatic immunity." Each of the agents has his own list of cover names, so the agency can't just hand out signed documents. The agents do not want to send their cover names to the agency; the enemy might have corrupted the agency's computer. On the other hand, the agency doesn't want to blindly sign any document an agent gives it. A clever agent might substitute a message like: "Agent (name) has retired and collects a million-dollar-a-year pension. Signed, Mr. President." In this case, blind signatures could be useful.

Assume that all the agents have 10 possible cover names, which they have chosen themselves and which no one else knows. Also assume that the agents don't care

under which cover name they are going to get diplomatic immunity. Also assume that the agency's computer is the Agency's Large Intelligent Computing Engine, or ALICE, and that our particular agent is the Bogota Operations Branch: BOB.

(1) BOB prepares n documents, each using a different cover name, giving himself diplomatic immunity.
(2) BOB blinds each of these documents with a different blinding factor.
(3) BOB sends the n blinded documents to ALICE.
(4) ALICE chooses $n - 1$ documents at random and asks BOB for the blinding factors for each of those documents.
(5) BOB sends ALICE the appropriate blinding factors.
(6) ALICE opens (i.e., she removes the blinding factor) $n - 1$ documents and makes sure they are correct—and not pension authorizations.
(7) ALICE signs the remaining document and sends it to BOB.
(8) Agent removes the blinding factor and reads his new cover name: "The Crimson Streak." The signed document gives him diplomatic immunity under that name.

This protocol is secure against BOB cheating. For him to cheat, he would have to predict accurately which document ALICE would not examine. The odds of him doing this are 1 in n—not very good. ALICE knows this and feels confident signing a document that she is not able to examine. With this one document, the protocol is the same as the previous completely blinded signature protocol and maintains all of its properties of anonymity.

There is a trick that makes BOB's chance of cheating even smaller. In step (4), ALICE randomly chooses $n/2$ of the documents to challenge, and BOB sends her the appropriate blinding factors in step (5). In step (7), ALICE multiplies together all of the unchallenged documents and signs the mega-document. In step (8), BOB strips off all the blinding factors. ALICE's signature is acceptable only if it is a valid signature of the product of $n/2$ identical documents. To cheat BOB has to be able to guess exactly which subset ALICE will challenge; the odds are much smaller than the odds of guessing which one document ALICE won't challenge.

BOB has another way to cheat. He can generate two different documents, one that ALICE is willing to sign and one that ALICE is not. Then he can find two different blinding factors that transform each document into the same blinded document. That way, if ALICE asks to examine the document, BOB gives her the blinding factor that transforms it into the benign document. If ALICE doesn't ask to see the document and signs it, he uses the blinding factor that transforms it into the malevolent document. While this is theoretically possible, the mathematics of the particular algorithms involved make the odds of BOB's being able to find such a pair negligibly small. In fact, it can be made as small as the odds of Bob being able to produce the signature on an arbitrary message himself. This issue is discussed further in Section 23.12.

(7) The merchant asks Alice to write a random identity string on the money order.

(8) Alice complies.

(9) The merchant takes the money order to the bank.

(10) The bank verifies the signature and checks its database to make sure a money order with the same uniqueness string has not been previously deposited. If it hasn't, the bank credits $1000 to the merchant's account. The bank records the uniqueness string and the identity string in a database.

(11) If the uniqueness string is in the database, the bank refuses to accept the money order. Then, it compares the identity string on the money order with the one stored in the database. If it is the same, the bank knows that the merchant photocopied the money order. If it is different, the bank knows that the person who bought the money order photocopied it.

This protocol assumes that the merchant cannot change the identity string once Alice writes it on the money order. The money order might have a series of little squares, which the merchant would require Alice to fill in with either Xs or Os. The money order might be made out of paper that tears if erased.

Since the interaction between the merchant and the bank takes place after Alice spends the money, the merchant could be stuck with a bad money order. Practical implementations of this protocol might require Alice to wait near the cash register during the merchant-bank interaction, much the same way as credit-card purchases are handled today.

Alice could also frame the merchant. She could spend a copy of the money order a second time, giving the same identity string in step (7). Unless the merchant keeps a database of money orders it already received, he would be fooled. The next protocol eliminates that problem.

Protocol #4

If it turns out that the person who bought the money order tried to cheat the merchant, the bank would want to know who that person was. To do that requires moving away from a physical analogy and into the world of cryptography.

The technique of secret splitting can be used to hide Alice's name in the digital money order.

(1) Alice prepares n anonymous money orders for a given amount.

Each of the money orders contains a different random uniqueness string, X, one long enough to make the chance of two being identical negligible.

On each money order, there are also n pairs of identity bit strings, I_1, I_2, . . . , I_n. (Yes, that's n different pairs on *each* check.) Each of these pairs is generated as follows: Alice creates a string that gives her name, address, and any other piece of identifying information that the bank wants to see. Then, she splits it into two pieces using the secret splitting protocol (see Section 3.6). Then, she commits to each piece using a bit-commitment protocol.

For example, I_{37} consists of two parts: I_{37_L} and I_{37_R}. Each part is a bit-committed packet that Alice can be asked to open and whose proper opening can be instantly verified. Any pair (e.g., I_{37_L} and I_{37_R}, but not I_{37_L} and I_{38_R}), reveals Alice's identity.

Each of the money orders looks like this:

```
Amount
Uniqueness String: X
Identity Strings:  I1 = (I1L, I1R)
                   I2 = (I2L, I2R)
                   ....
                   In = (InL, InR)
```

(2) Alice blinds all n money orders, using a blind signature protocol. She gives them all to the bank.

(3) The bank asks Alice to unblind $n - 1$ of the money orders at random and confirms that they are all well formed. The bank checks the amount, the uniqueness string, and asks Alice to reveal all of the identity strings.

(4) If the bank is satisfied that Alice did not make any attempts to cheat, it signs the one remaining blinded money order. The bank hands the blinded money order back to Alice and deducts the amount from her account.

(5) Alice unblinds the money order and spends it with a merchant.

(6) The merchant verifies the bank's signature to make sure the money order is legitimate.

(7) The merchant asks Alice to randomly reveal either the left half or the right half of each identity string on the money order. In effect, the merchant gives Alice a random n-bit **selector string**, $b_1, b_2, \ldots, b_n$. Alice opens either the left or right half of I_i, depending on whether b_i is a 0 or a 1.

(8) Alice complies.

(9) The merchant takes the money order to the bank.

(10) The bank verifies the signature and checks its database to make sure a money order with the same uniqueness string has not been previously deposited. If it hasn't, the bank credits the amount to the merchant's account. The bank records the uniqueness string and all of the identity information in a database.

(11) If the uniqueness string is in the database, the bank refuses to accept the money order. Then, it compares the identity string on the money order with the one stored in the database. If it is the same, the bank knows that the merchant copied the money order. If it is different, the bank knows that the person who bought the money order photocopied it. Since the second merchant who accepted the money order handed Alice a different selector string than did the first merchant, the bank finds a bit position where one merchant had Alice open the left half and the other merchant had Alice open the right half. The bank XORs the two halves together to reveal Alice's identity.

This is quite an amazing protocol, so let's look at it from various angles.

Can Alice cheat? Her digital money order is nothing more than a string of bits, so she can copy it. Spending it the first time won't be a problem; she'll just complete the protocol and everything will go smoothly. The merchant will give her a random n-bit selector string in step (7) and Alice will open either the left half or right half of each I_i in step (8). In step (10), the bank will record all of this data, as well as the money order's uniqueness string.

When she tries to use the same digital money order a second time, the merchant (either the same merchant or a different merchant) will give her a different random selector string in step (7). Alice must comply in step (8); not doing so will immediately alert the merchant that something is suspicious. Now, when the merchant brings the money order to the bank in step (10), the bank would immediately notice that a money order with the same uniqueness string was already deposited. The bank then compares the opened halves of the identity strings. The odds that the two random selector strings are the same is 1 in 2^n; it isn't likely to happen before the next ice age. Now, the bank finds a pair with one half opened the first time and the other half opened the second time. It XORs the two halves together, and out pops Alice's name. The bank knows who tried to spend the money order twice.

Note that this protocol doesn't keep Alice from trying to cheat; it detects her cheating with almost certainty. Alice can't prevent her identity from being revealed if she cheats. She can't change either the uniqueness string or any of the identity strings, because then the bank's signature will no longer be valid. The merchant will immediately notice that in step (6).

Alice could try to sneak a bad money order past the bank, one on which the identity strings don't reveal her name; or better yet, one whose identity strings reveal someone else's name. The odds of her getting this ruse past the bank in step (3) are 1 in n. These aren't impossible odds, but if you make the penalty severe enough, Alice won't try it. Or, you could increase the number of redundant money orders that Alice makes in step (1).

Can the merchant cheat? His chances are even worse. He can't deposit the money order twice; the bank will notice the repeated use of the selector string. He can't fake blaming Alice; only she can open any of the identity strings.

Even collusion between Alice and the merchant can't cheat the bank. As long as the bank signs the money order with the uniqueness string, the bank is assured of only having to make good on the money order once.

What about the bank? Can it figure out that the money order it accepted from the merchant was the one it signed for Alice? Alice is protected by the blind signature protocol in steps (2) through (5). The bank cannot make the connection, even if it keeps complete records of every transaction. Even more strongly, there is no way for the bank and the merchant to get together to figure out who Alice is. Alice can walk in the store and, completely anonymously, make her purchase.

Eve can cheat. If she can eavesdrop on the communication between Alice and the merchant, and if she can get to the bank before the merchant does, she can deposit the digital cash first. The bank will accept it and, even worse, when the merchant tries to deposit the cash he will be identified as a cheater. If Eve steals and spends

Alice's cash before Alice can, then Alice will be identified as a cheater. There's no way to prevent this; it is a direct result of the anonynimity of the cash. Both Alice and the merchant have to protect their bits as they would paper money.

This protocol lies somewhere between an arbitrated protocol and a self-enforcing protocol. Both Alice and the merchant trust the bank to make good on the money orders, but Alice does not have to trust the bank with knowledge of her purchases.

Digital Cash and the Perfect Crime

Digital cash has its dark side, too. Sometimes people don't want so much privacy. Watch Alice commit the perfect crime [1575]:

(1) Alice kidnaps a baby.

(2) Alice prepares 10,000 anonymous money orders for $1000 (or as many as she wants for whatever denomination she wants).

(3) Alice blinds all 10,000 money orders, using a blind signature protocol. She sends them to the authorities with the threat to kill the baby unless the following instructions are met:

   (a) Have a bank sign all 10,000 money orders.

   (b) Publish the results in a newspaper.

(4) The authorities comply.

(5) Alice buys a newspaper, unblinds the money orders, and starts spending them. There is no way for the authorities to trace the money orders to her.

(6) Alice frees the baby.

Note that this situation is much worse than any involving physical tokens—cash, for example. Without physical contact, the police have less opportunity to apprehend the kidnapper.

In general, though, digital cash isn't a good deal for criminals. The problem is that the anonymity only works one way: The spender is anonymous, but the merchant is not. Moreover, the merchant cannot hide the fact that he received money. Digital cash will make it easy for the government to determine how much money you made, but impossible to determine what you spent it on.

Practical Digital Cash

A Dutch company, DigiCash, owns most of the digital cash patents and has implemented digital cash protocols in working products. Anyone interested should contact DigiCash BV, Kruislaan 419, 1098 VA Amsterdam, Netherlands.

Other Digital Cash Protocols

There are other digital cash protocols; see [707,1554,734,1633,973]. Some of them involve some pretty complicated mathematics. Generally, the various digital cash protocols can be divided into various categories. **On-line** systems require the merchant to communicate with the bank at every sale, much like today's

(1) Alice and Bob agree on a random k and an m such that

$$km \equiv e \pmod{n}$$

They should choose the numbers randomly, using a coin-flip protocol to generate k and then computing m. If both k and m are greater than 3, the protocol continues. Otherwise, they choose again.

(2) Alice and Bob generate a random ciphertext, C. Again, they should use a coin-flip protocol.

(3) Alice, using Carol's private key, computes

$$M = C^d \bmod n$$

She then computes

$$X = M^k \bmod n$$

and sends X to Bob.

(4) Bob confirms that $X^m \bmod n = C$. If it does, he believes Alice.

A similar protocol can be used to demonstrate the ability to break a discrete logarithm problem [888].

Zero-Knowledge Proof that n Is a Blum Integer

There are no known truly practical zero-knowledge proofs that $n = pq$, where p and q are primes congruent to 3 modulo 4. However, if you allow n to be of the form $p^r q^s$, where r and s are odd, then the properties which make Blum integers useful in cryptography still hold. And there exists a zero-knowledge proof that n is of that form.

Assume Alice knows the factorization of the Blum integer n, where n is of the form previously discussed. Here's how she can prove to Bob that n is of that form [660].

(1) Alice sends Bob a number u which has a Jacobi symbol -1 modulo n.

(2) Alice and Bob jointly agree on random bits: $b_1, b_2, \ldots, b_k$.

(3) Alice and Bob jointly agree on random numbers: $x_1, x_2, \ldots, x_k$.

(4) For each $i = 1, 2, \ldots, k$, Alice sends Bob a square root modulo n, of one of the four numbers: $x_i, -x_i, ux_i, -ux_i$. The square root must have the Jacobi symbol b_i.

The odds of Alice successfully cheating are one in 2^k.

23.12 Blind Signatures

The notion of blind signatures (see Section 5.3) was invented by David Chaum [317,323], who also invented their first implementation [318]. It uses the RSA algorithm.

Bob has a public key, e, a private key, d, and a public modulus, n. Alice wants Bob to sign message m blindly.

(1) Alice chooses a random value, k, between 1 and n. Then she blinds m by computing

$$t = mk^e \bmod n$$

(2) Bob signs t

$$t^d = (mk^e)^d \bmod n$$

(3) Alice unblinds t^d by computing

$$s = t^d/k \bmod n$$

(4) And the result is

$$s = m^d \bmod n$$

This can easily be shown

$$t^d \equiv (mk^e)^d \equiv m^d k \pmod{n}, \text{ so } t^d/k = m^d k/k \equiv m^d \pmod{n}.$$

Chaum invented a family of more complicated blind signature algorithms in [320,324], called blind unanticipated signatures. These signatures are more complex in construction, but more flexible.

23.13 Oblivious Transfer

In this protocol by Michael Rabin [1286], Alice has a 50 percent chance of sending Bob two primes, p, and q. Alice will not know whether the transfer is successful. (See Section 5.5.) (This protocol can be used to send Bob any message with a 50 percent success rate if p and q reveal an RSA private key.)

(1) Alice sends Bob the product of the two primes: $n = pq$.

(2) Bob chooses a random x less than n, such that x is relatively prime to n. He sends Alice:

$$a = x^2 \bmod n$$

(3) Alice, knowing p and q, computes the four roots of a: x, $n - x$, y, and $n - y$. She chooses one of these roots at random and sends it to Bob.

(4) If Bob receives y or $n - y$, he can compute the greatest common divisor of $x + y$ and n, which is either p or q. Then, of course, $n/p = q$.
If Bob receives x or $n - x$, he can't compute anything.

This protocol may have a weakness: It might be the case that Bob can compute a number a such that given the square root of a you can calculate a factor of n all the time.