Bitcoin: A Peer-to-Peer Electronic Cash System

Satoshi Nakamoto
satoshin@gmx.com
www.bitcoin.org

Abstract. A purely peer-to-peer version of electronic cash would allow online payments to be sent directly from one party to another without going through a financial institution. Digital signatures provide part of the solution, but the main benefits are lost if a trusted third party is still required to prevent double-spending. We propose a solution to the double-spending problem using a peer-to-peer network. The network timestamps transactions by hashing them into an ongoing chain of hash-based proof-of-work, forming a record that cannot be changed without redoing the proof-of-work. The longest chain not only serves as proof of the sequence of events witnessed, but proof that it came from the largest pool of CPU power. As long as a majority of CPU power is controlled by nodes that are not cooperating to attack the network, they'll generate the longest chain and outpace attackers. The network itself requires minimal structure. Messages are broadcast on a best effort basis, and nodes can leave and rejoin the network at will, accepting the longest proof-of-work chain as proof of what happened while they were gone.

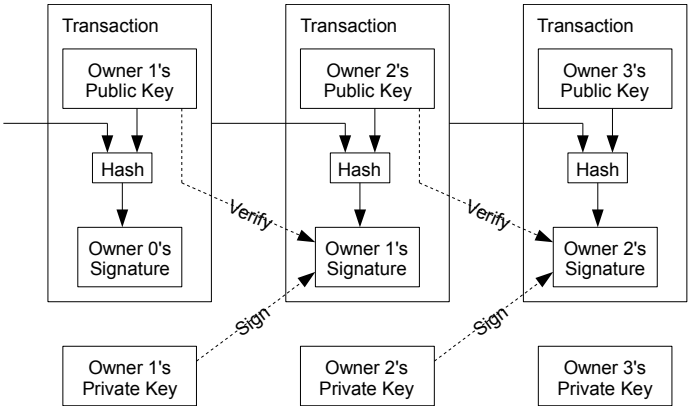
1. Introduction

Commerce on the Internet has come to rely almost exclusively on financial institutions serving as trusted third parties to process electronic payments. While the system works well enough for most transactions, it still suffers from the inherent weaknesses of the trust based model. Completely non-reversible transactions are not really possible, since financial institutions cannot avoid mediating disputes. The cost of mediation increases transaction costs, limiting the minimum practical transaction size and cutting off the possibility for small casual transactions, and there is a broader cost in the loss of ability to make non-reversible payments for non-reversible services. With the possibility of reversal, the need for trust spreads. Merchants must be wary of their customers, hassling them for more information than they would otherwise need. A certain percentage of fraud is accepted as unavoidable. These costs and payment uncertainties can be avoided in person by using physical currency, but no mechanism exists to make payments over a communications channel without a trusted party.

What is needed is an electronic payment system based on cryptographic proof instead of trust, allowing any two willing parties to transact directly with each other without the need for a trusted third party. Transactions that are computationally impractical to reverse would protect sellers from fraud, and routine escrow mechanisms could easily be implemented to protect buyers. In this paper, we propose a solution to the double-spending problem using a peer-to-peer distributed timestamp server to generate computational proof of the chronological order of transactions. The system is secure as long as honest nodes collectively control more CPU power than any cooperating group of attacker nodes.

2. Transactions

We define an electronic coin as a chain of digital signatures. Each owner transfers the coin to the next by digitally signing a hash of the previous transaction and the public key of the next owner and adding these to the end of the coin. A payee can verify the signatures to verify the chain of ownership.

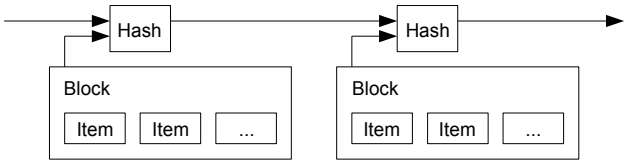


The problem of course is the payee can't verify that one of the owners did not double-spend the coin. A common solution is to introduce a trusted central authority, or mint, that checks every transaction for double spending. After each transaction, the coin must be returned to the mint to issue a new coin, and only coins issued directly from the mint are trusted not to be double-spent. The problem with this solution is that the fate of the entire money system depends on the company running the mint, with every transaction having to go through them, just like a bank.

We need a way for the payee to know that the previous owners did not sign any earlier transactions. For our purposes, the earliest transaction is the one that counts, so we don't care about later attempts to double-spend. The only way to confirm the absence of a transaction is to be aware of all transactions. In the mint based model, the mint was aware of all transactions and decided which arrived first. To accomplish this without a trusted party, transactions must be publicly announced [1], and we need a system for participants to agree on a single history of the order in which they were received. The payee needs proof that at the time of each transaction, the majority of nodes agreed it was the first received.

3. Timestamp Server

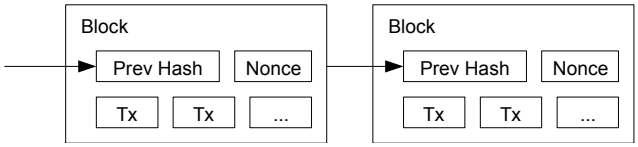
The solution we propose begins with a timestamp server. A timestamp server works by taking a hash of a block of items to be timestamped and widely publishing the hash, such as in a newspaper or Usenet post [2-5]. The timestamp proves that the data must have existed at the time, obviously, in order to get into the hash. Each timestamp includes the previous timestamp in its hash, forming a chain, with each additional timestamp reinforcing the ones before it.



4. Proof-of-Work

To implement a distributed timestamp server on a peer-to-peer basis, we will need to use a proof-of-work system similar to Adam Back's Hashcash [6], rather than newspaper or Usenet posts. The proof-of-work involves scanning for a value that when hashed, such as with SHA-256, the hash begins with a number of zero bits. The average work required is exponential in the number of zero bits required and can be verified by executing a single hash.

For our timestamp network, we implement the proof-of-work by incrementing a nonce in the block until a value is found that gives the block's hash the required zero bits. Once the CPU effort has been expended to make it satisfy the proof-of-work, the block cannot be changed without redoing the work. As later blocks are chained after it, the work to change the block would include redoing all the blocks after it.



The proof-of-work also solves the problem of determining representation in majority decision making. If the majority were based on one-IP-address-one-vote, it could be subverted by anyone able to allocate many IPs. Proof-of-work is essentially one-CPU-one-vote. The majority decision is represented by the longest chain, which has the greatest proof-of-work effort invested in it. If a majority of CPU power is controlled by honest nodes, the honest chain will grow the fastest and outpace any competing chains. To modify a past block, an attacker would have to redo the proof-of-work of the block and all blocks after it and then catch up with and surpass the work of the honest nodes. We will show later that the probability of a slower attacker catching up diminishes exponentially as subsequent blocks are added.

To compensate for increasing hardware speed and varying interest in running nodes over time, the proof-of-work difficulty is determined by a moving average targeting an average number of blocks per hour. If they're generated too fast, the difficulty increases.

5. Network

The steps to run the network are as follows:

1) New transactions are broadcast to all nodes.
2) Each node collects new transactions into a block.
3) Each node works on finding a difficult proof-of-work for its block.
4) When a node finds a proof-of-work, it broadcasts the block to all nodes.
5) Nodes accept the block only if all transactions in it are valid and not already spent.
6) Nodes express their acceptance of the block by working on creating the next block in the chain, using the hash of the accepted block as the previous hash.

Nodes always consider the longest chain to be the correct one and will keep working on extending it. If two nodes broadcast different versions of the next block simultaneously, some nodes may receive one or the other first. In that case, they work on the first one they received, but save the other branch in case it becomes longer. The tie will be broken when the next proof-of-work is found and one branch becomes longer; the nodes that were working on the other branch will then switch to the longer one.

New transaction broadcasts do not necessarily need to reach all nodes. As long as they reach many nodes, they will get into a block before long. Block broadcasts are also tolerant of dropped messages. If a node does not receive a block, it will request it when it receives the next block and realizes it missed one.

6. Incentive

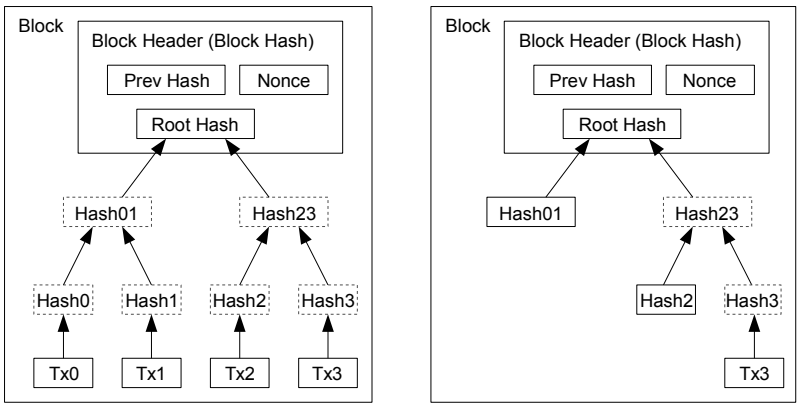
By convention, the first transaction in a block is a special transaction that starts a new coin owned by the creator of the block. This adds an incentive for nodes to support the network, and provides a way to initially distribute coins into circulation, since there is no central authority to issue them. The steady addition of a constant of amount of new coins is analogous to gold miners expending resources to add gold to circulation. In our case, it is CPU time and electricity that is expended.

The incentive can also be funded with transaction fees. If the output value of a transaction is less than its input value, the difference is a transaction fee that is added to the incentive value of the block containing the transaction. Once a predetermined number of coins have entered circulation, the incentive can transition entirely to transaction fees and be completely inflation free.

The incentive may help encourage nodes to stay honest. If a greedy attacker is able to assemble more CPU power than all the honest nodes, he would have to choose between using it to defraud people by stealing back his payments, or using it to generate new coins. He ought to find it more profitable to play by the rules, such rules that favour him with more new coins than everyone else combined, than to undermine the system and the validity of his own wealth.

7. Reclaiming Disk Space

Once the latest transaction in a coin is buried under enough blocks, the spent transactions before it can be discarded to save disk space. To facilitate this without breaking the block's hash, transactions are hashed in a Merkle Tree [7][2][5], with only the root included in the block's hash. Old blocks can then be compacted by stubbing off branches of the tree. The interior hashes do not need to be stored.



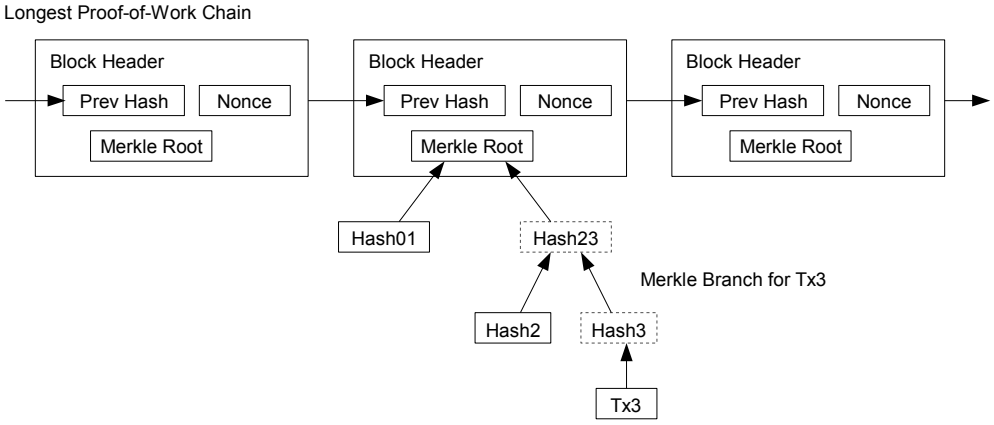
Transactions Hashed in a Merkle Tree

After Pruning Tx0-2 from the Block

A block header with no transactions would be about 80 bytes. If we suppose blocks are generated every 10 minutes, 80 bytes * 6 * 24 * 365 = 4.2MB per year. With computer systems typically selling with 2GB of RAM as of 2008, and Moore's Law predicting current growth of 1.2GB per year, storage should not be a problem even if the block headers must be kept in memory.

8. Simplified Payment Verification

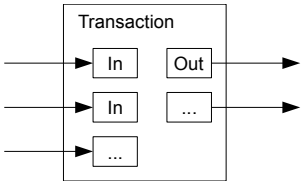
It is possible to verify payments without running a full network node. A user only needs to keep a copy of the block headers of the longest proof-of-work chain, which he can get by querying network nodes until he's convinced he has the longest chain, and obtain the Merkle branch linking the transaction to the block it's timestamped in. He can't check the transaction for himself, but by linking it to a place in the chain, he can see that a network node has accepted it, and blocks added after it further confirm the network has accepted it.



As such, the verification is reliable as long as honest nodes control the network, but is more vulnerable if the network is overpowered by an attacker. While network nodes can verify transactions for themselves, the simplified method can be fooled by an attacker's fabricated transactions for as long as the attacker can continue to overpower the network. One strategy to protect against this would be to accept alerts from network nodes when they detect an invalid block, prompting the user's software to download the full block and alerted transactions to confirm the inconsistency. Businesses that receive frequent payments will probably still want to run their own nodes for more independent security and quicker verification.

9. Combining and Splitting Value

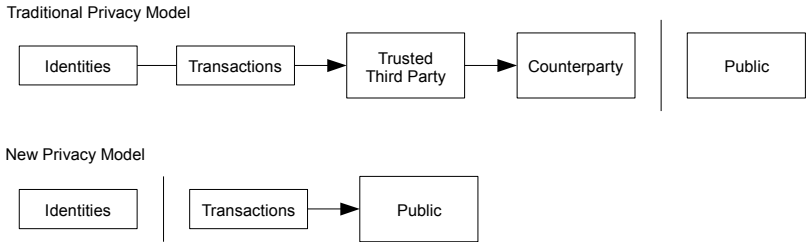
Although it would be possible to handle coins individually, it would be unwieldy to make a separate transaction for every cent in a transfer. To allow value to be split and combined, transactions contain multiple inputs and outputs. Normally there will be either a single input from a larger previous transaction or multiple inputs combining smaller amounts, and at most two outputs: one for the payment, and one returning the change, if any, back to the sender.



It should be noted that fan-out, where a transaction depends on several transactions, and those transactions depend on many more, is not a problem here. There is never the need to extract a complete standalone copy of a transaction's history.

10. Privacy

The traditional banking model achieves a level of privacy by limiting access to information to the parties involved and the trusted third party. The necessity to announce all transactions publicly precludes this method, but privacy can still be maintained by breaking the flow of information in another place: by keeping public keys anonymous. The public can see that someone is sending an amount to someone else, but without information linking the transaction to anyone. This is similar to the level of information released by stock exchanges, where the time and size of individual trades, the "tape", is made public, but without telling who the parties were.



As an additional firewall, a new key pair should be used for each transaction to keep them from being linked to a common owner. Some linking is still unavoidable with multi-input transactions, which necessarily reveal that their inputs were owned by the same owner. The risk is that if the owner of a key is revealed, linking could reveal other transactions that belonged to the same owner.

11. Calculations

We consider the scenario of an attacker trying to generate an alternate chain faster than the honest chain. Even if this is accomplished, it does not throw the system open to arbitrary changes, such as creating value out of thin air or taking money that never belonged to the attacker. Nodes are not going to accept an invalid transaction as payment, and honest nodes will never accept a block containing them. An attacker can only try to change one of his own transactions to take back money he recently spent.

The race between the honest chain and an attacker chain can be characterized as a Binomial Random Walk. The success event is the honest chain being extended by one block, increasing its lead by +1, and the failure event is the attacker's chain being extended by one block, reducing the gap by -1.

The probability of an attacker catching up from a given deficit is analogous to a Gambler's Ruin problem. Suppose a gambler with unlimited credit starts at a deficit and plays potentially an infinite number of trials to try to reach breakeven. We can calculate the probability he ever reaches breakeven, or that an attacker ever catches up with the honest chain, as follows [8]:

p = probability an honest node finds the next block
q = probability the attacker finds the next block
q_z = probability the attacker will ever catch up from z blocks behind

$$q_z = \begin{cases} 1 & \text{if } p \leq q \\ (q/p)^z & \text{if } p > q \end{cases}$$

Given our assumption that $p > q$, the probability drops exponentially as the number of blocks the attacker has to catch up with increases. With the odds against him, if he doesn't make a lucky lunge forward early on, his chances become vanishingly small as he falls further behind.

We now consider how long the recipient of a new transaction needs to wait before being sufficiently certain the sender can't change the transaction. We assume the sender is an attacker who wants to make the recipient believe he paid him for a while, then switch it to pay back to himself after some time has passed. The receiver will be alerted when that happens, but the sender hopes it will be too late.

The receiver generates a new key pair and gives the public key to the sender shortly before signing. This prevents the sender from preparing a chain of blocks ahead of time by working on it continuously until he is lucky enough to get far enough ahead, then executing the transaction at that moment. Once the transaction is sent, the dishonest sender starts working in secret on a parallel chain containing an alternate version of his transaction.

The recipient waits until the transaction has been added to a block and z blocks have been linked after it. He doesn't know the exact amount of progress the attacker has made, but assuming the honest blocks took the average expected time per block, the attacker's potential progress will be a Poisson distribution with expected value:

$$\lambda = z\frac{q}{p}$$

To get the probability the attacker could still catch up now, we multiply the Poisson density for each amount of progress he could have made by the probability he could catch up from that point:

$$\sum_{k=0}^{\infty} \frac{\lambda^k e^{-\lambda}}{k!} \cdot \begin{cases} (q/p)^{(z-k)} & \text{if } k \le z \\ 1 & \text{if } k > z \end{cases}$$

Rearranging to avoid summing the infinite tail of the distribution...

$$1 - \sum_{k=0}^{z} \frac{\lambda^k e^{-\lambda}}{k!} \left(1 - (q/p)^{(z-k)}\right)$$

Converting to C code...

```
#include <math.h>
double AttackerSuccessProbability(double q, int z)
{
    double p = 1.0 - q;
    double lambda = z * (q / p);
    double sum = 1.0;
    int i, k;
    for (k = 0; k <= z; k++)
    {
        double poisson = exp(-lambda);
        for (i = 1; i <= k; i++)
            poisson *= lambda / i;
        sum -= poisson * (1 - pow(q / p, z - k));
    }
    return sum;
}
```

Running some results, we can see the probability drop off exponentially with z.

```
q=0.1
z=0     P=1.0000000
z=1     P=0.2045873
z=2     P=0.0509779
z=3     P=0.0131722
z=4     P=0.0034552
z=5     P=0.0009137
z=6     P=0.0002428
z=7     P=0.0000647
z=8     P=0.0000173
z=9     P=0.0000046
z=10    P=0.0000012

q=0.3
z=0     P=1.0000000
z=5     P=0.1773523
z=10    P=0.0416605
z=15    P=0.0101008
z=20    P=0.0024804
z=25    P=0.0006132
z=30    P=0.0001522
z=35    P=0.0000379
z=40    P=0.0000095
z=45    P=0.0000024
z=50    P=0.0000006
```

Solving for P less than 0.1%...

```
P < 0.001
q=0.10   z=5
q=0.15   z=8
q=0.20   z=11
q=0.25   z=15
q=0.30   z=24
q=0.35   z=41
q=0.40   z=89
q=0.45   z=340
```

12. Conclusion

We have proposed a system for electronic transactions without relying on trust. We started with the usual framework of coins made from digital signatures, which provides strong control of ownership, but is incomplete without a way to prevent double-spending. To solve this, we proposed a peer-to-peer network using proof-of-work to record a public history of transactions that quickly becomes computationally impractical for an attacker to change if honest nodes control a majority of CPU power. The network is robust in its unstructured simplicity. Nodes work all at once with little coordination. They do not need to be identified, since messages are not routed to any particular place and only need to be delivered on a best effort basis. Nodes can leave and rejoin the network at will, accepting the proof-of-work chain as proof of what happened while they were gone. They vote with their CPU power, expressing their acceptance of valid blocks by working on extending them and rejecting invalid blocks by refusing to work on them. Any needed rules and incentives can be enforced with this consensus mechanism.

References

[1] W. Dai, "b-money," http://www.weidai.com/bmoney.txt, 1998.

[2] H. Massias, X.S. Avila, and J.-J. Quisquater, "Design of a secure timestamping service with minimal trust requirements," In *20th Symposium on Information Theory in the Benelux*, May 1999.

[3] S. Haber, W.S. Stornetta, "How to time-stamp a digital document," In *Journal of Cryptology*, vol 3, no 2, pages 99-111, 1991.

[4] D. Bayer, S. Haber, W.S. Stornetta, "Improving the efficiency and reliability of digital time-stamping," In *Sequences II: Methods in Communication, Security and Computer Science*, pages 329-334, 1993.

[5] S. Haber, W.S. Stornetta, "Secure names for bit-strings," In *Proceedings of the 4th ACM Conference on Computer and Communications Security*, pages 28-35, April 1997.

[6] A. Back, "Hashcash - a denial of service counter-measure," http://www.hashcash.org/papers/hashcash.pdf, 2002.

[7] R.C. Merkle, "Protocols for public key cryptosystems," In *Proc. 1980 Symposium on Security and Privacy*, IEEE Computer Society, pages 122-133, April 1980.

[8] W. Feller, "An introduction to probability theory and its applications," 1957.

I am fascinated by Tim May's crypto-anarchy. Unlike the communities traditionally associated with the word "anarchy", in a crypto-anarchy the government is not temporarily destroyed but permanently forbidden and permanently unnecessary. It's a community where the threat of violence is impotent because violence is impossible, and violence is impossible because its participants cannot be linked to their true names or physical locations.

Until now it's not clear, even theoretically, how such a community could operate. A community is defined by the cooperation of its participants, and efficient cooperation requires a medium of exchange (money) and a way to enforce contracts. Traditionally these services have been provided by the government or government sponsored institutions and only to legal entities. In this article I describe a protocol by which these services can be provided to and by untraceable entities.

I will actually describe two protocols. The first one is impractical, because it makes heavy use of a synchronous and unjammable anonymous broadcast channel. However it will motivate the second, more practical protocol. In both cases I will assume the existence of an untraceable network, where senders and receivers are identified only by digital pseudonyms (i.e. public keys) and every messages is signed by its sender and encrypted to its receiver.

In the first protocol, every participant maintains a (seperate) database of how much money belongs to each pseudonym. These accounts collectively define the ownership of money, and how these accounts are updated is the subject of this protocol.

1. The creation of money. Anyone can create money by broadcasting the solution to a previously unsolved computational problem. The only conditions are that it must be easy to determine how much computing effort it took to solve the problem and the solution must otherwise have no value, either practical or intellectual. The number of monetary units created is equal to the cost of the computing effort in terms of a standard basket of commodities. For example if a problem takes 100 hours to solve on the computer that solves it most economically, and it takes 3 standard baskets to purchase 100 hours of computing time on that computer on the open market, then upon the broadcast of the solution to that problem everyone credits the broadcaster's account by 3 units.

2. The transfer of money. If Alice (owner of pseudonym K_A) wishes to transfer X units of money to Bob (owner of pseudonym K_B), she broadcasts the message "I give X units of money to K_B" signed by K_A. Upon the broadcast of this message, everyone debits K_A's account by X units and credits K_B's account by X units, unless this would create a negative balance in K_A's account in which case the message is ignored.

3. The effecting of contracts. A valid contract must include a maximum reparation in case of default for each participant party to it. It should also include a party who will perform arbitration should there be a dispute. All parties to a contract including the arbitrator must broadcast their signatures of it before it becomes effective. Upon the broadcast of the contract and all signatures, every participant debits the account of each party by the amount of his maximum reparation and credits a special account identified by a secure hash of the contract by the sum the maximum

reparations. The contract becomes effective if the debits succeed for every party without producing a negative balance, otherwise the contract is ignored and the accounts are rolled back. A sample contract might look like this:

K_A agrees to send K_B the solution to problem P before 0:0:0 1/1/2000. K_B agrees to pay K_A 100 MU (monetary units) before 0:0:0 1/1/2000. K_C agrees to perform arbitration in case of dispute. K_A agrees to pay a maximum of 1000 MU in case of default. K_B agrees to pay a maximum of 200 MU in case of default. K_C agrees to pay a maximum of 500 MU in case of default.

4. The conclusion of contracts. If a contract concludes without dispute, each party broadcasts a signed message "The contract with SHA-1 hash H concludes without reparations." or possibly "The contract with SHA-1 hash H concludes with the following reparations: ..." Upon the broadcast of all signatures, every participant credits the account of each party by the amount of his maximum reparation, removes the contract account, then credits or debits the account of each party according to the reparation schedule if there is one.

5. The enforcement of contracts. If the parties to a contract cannot agree on an appropriate conclusion even with the help of the arbitrator, each party broadcasts a suggested reparation/fine schedule and any arguments or evidence in his favor. Each participant makes a determination as to the actual reparations and/or fines, and modifies his accounts accordingly.

In the second protocol, the accounts of who has how much money are kept by a subset of the participants (called servers from now on) instead of everyone. These servers are linked by a Usenet-style broadcast channel. The format of transaction messages broadcasted on this channel remain the same as in the first protocol, but the affected participants of each transaction should verify that the message has been received and successfully processed by a randomly selected subset of the servers.

Since the servers must be trusted to a degree, some mechanism is needed to keep them honest. Each server is required to deposit a certain amount of money in a special account to be used as potential fines or rewards for proof of misconduct. Also, each server must periodically publish and commit to its current money creation and money ownership databases. Each participant should verify that his own account balances are correct and that the sum of the account balances is not greater than the total amount of money created. This prevents the servers, even in total collusion, from permanently and costlessly expanding the money supply. New servers can also use the published databases to synchronize with existing servers.

The protocol proposed in this article allows untraceable pseudonymous entities to cooperate with each other more efficiently, by providing them with a medium of exchange and a method of enforcing contracts. The protocol can probably be made more efficient and secure, but I hope this is a step toward making crypto-anarchy a practical as well as theoretical possibility.

Appendix A: alternative b-money creation

One of the more problematic parts in the b-money protocol is money creation. This part of the protocol requires that all of the account keepers decide and agree on the cost of particular computations. Unfortunately because computing technology tends to advance rapidly and not always publicly, this information may be unavailable, inaccurate, or outdated, all of which would cause serious problems for the protocol.

So I propose an alternative money creation subprotocol, in which account keepers (everyone in the first protocol, or the servers in the second protocol) instead decide and agree on the amount of b-money to be created each period, with the cost of creating that money determined by an auction. Each money creation period is divided up into four phases, as follows:

1. Planning. The account keepers compute and negotiate with each other to determine an optimal increase in the money supply for the next period. Whether or not the account keepers can reach a consensus, they each broadcast their money creation quota and any macroeconomic calculations done to support the figures.

2. Bidding. Anyone who wants to create b-money broadcasts a bid in the form of <x, y> where x is the amount of b-money he wants to create, and y is an unsolved problem from a predetermined problem class. Each problem in this class should have a nominal cost (in MIPS-years say) which is publicly agreed on.

3. Computation. After seeing the bids, the ones who placed bids in the bidding phase may now solve the problems in their bids and broadcast the solutions.

4. Money creation. Each account keeper accepts the highest bids (among those who actually broadcasted solutions) in terms of nominal cost per unit of b-money created and credits the bidders' accounts accordingly.

Design Of A Secure Timestamping Service With Minimal Trust Requirement

Article · July 2002

Source: CiteSeer

CITATIONS
87

READS
2,759

3 authors, including:

Jean-Jacques Quisquater
Université Catholique de Louvain - UCLouvain
409 PUBLICATIONS **14,376** CITATIONS

SEE PROFILE

DESIGN OF A SECURE TIMESTAMPING SERVICE WITH MINIMAL TRUST REQUIREMENT

H. Massias, X. Serret Avila, J.-J. Quisquater
UCL Crypto group
Place du Levant, 3 , B-1348 Louvain-la-Neuve, Belgium
massias, serret, jjq@dice.ucl.ac.be

This paper presents our design of a timestamping system for the Belgian project TIMESEC. We first introduce the timestamping method used and we justify our choice for it. Then we present the design of our implementation as well as some of the important issues we found and the solutions we gave to them.

INTRODUCTION

The creation date of digital documents and the times expressed in them are becoming increasingly important as digital documents are being introduced into the legal domain.

We define "digital timestamp" as a digital certificate intended to assure the existence of a generic digital document at a certain time.

In order to produce fully trusted timestamps, very specific designs have been introduced. We give an overview of the most relevant methods and we introduce the one we used for the implementation of the Belgian project TIMESEC (see [PRQ+98]), justifying our choice for it. Then we present the design of the timestamping system we made for this project. We separate the different processes that are: document timestamping, timestamp verification, auditing, system start-up and system shutdown.

INTRODUCTION OF THE TIMESTAMPING TECHNIQUES

There are two families of timestamping techniques: those that work with a trusted third party and those that are based on the concept of distributed trust. Techniques based on a trusted third party rely on the impartiality of the entity that is in charge of issuing the timestamps. Techniques based on the distributed trust consist on making documents dated and signed by a large set of people in order to convince the verifiers that we could not have corrupted all of them. The trusted third party techniques can also be classified into two different kinds: those where the third party is completely trusted and those where it is partially

trusted. A detailed study of timestamping techniques can be found in [MQ97]. We believe that techniques based on distributed trust are not really workable in a professional environment, that is why we concentrate on the trusted third party approach. Nevertheless, we imposed to ourselves the requirement to lower the necessary trust on the third party to the maximum extend.

The "easy" solution, which consists on concatenating the document with the current time and sign the result, has been discarded because it has two main drawbacks:

1. We must completely trust the third party, called Secure Timestamp Authority (STA), which can issue undetectable back-dated timestamps.
2. The limited lifetime of cryptographic signatures, which can be shorter than the document time-to-life.

The timestamping method that we have chosen uses a binary tree structure and has been described in [HS91] and [HS97]. This method works by rounds. For each round a binary tree is constructed with the requests filled during it. The rounds have a fixed duration, which is the result of a trade-off between the timestamps accuracy and the number of requests submitted. In Figure 1 we can see a graphical representation of a round constructed using this method.

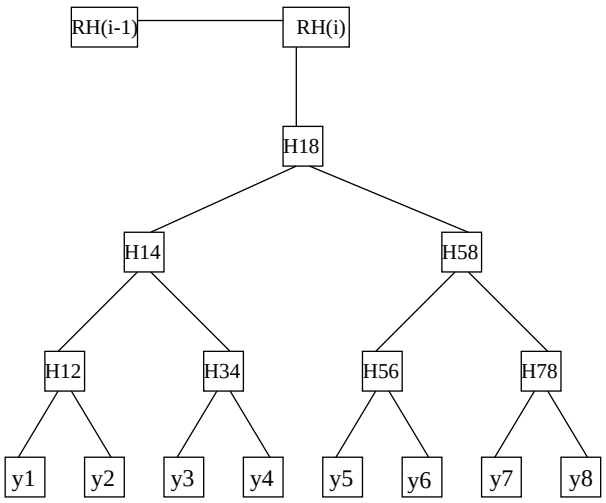


Figure 1: The binary tree structure

Each of the timestamp requests consists on a hash value of a given document. The leafs of the tree are each of those hash values. The leaf values are then

concatenated by two and hashed again to obtain the parent value (Ex: $H_{34} = H(y_3 \mid y_4)$). The process is repeated for each level until a single value is obtained. Finally, the top value of the round tree (H_{18}), called the "Round Root Value", is then concatenated with the value obtained for the preceding round (RH_{i-1}) and then hashed again to obtain the actual "Round Value" (RH_i).

The timestamp of the document contains all the values necessary to rebuilt the corresponding branch of the tree. For example, the timestamp for y_4 contains $\{(y_3, L), (H_{12}, L), (H_{58}, R), (RH_{i-1}, L)\}$. The verification process consists of re-building the tree's branch and the linking chain of "Round Values" until a trusted (from the verifier point of view) "Round Value" is recomputed. This verification method is explained in detail in [HS91] and [MQ97].

Periodically, one of the "Round Values" is published on an unmodifiable, widely witnessed media (Ex: newspaper...). These special "Round Values", which we will call "Big Round Values", are the base of the trust for all the timestamps issued. All verifiers must trust these "Big Round Values" as well as the time associated with them. This is a reasonable requirement because those values are widely witnessed. The absolute time trusted by all the potential verifiers is the time indicated by the unmodifiable media. We suppose that this time is the same than the time indicated by the STA for the "Big Round". Forcing the clients to check the timestamps as soon as they get them is another requirement. In that way the process is continuously audited and the STA will not have any margin to maneuver in an untrusted way.

A very useful method for extending the lifetime of timestamps is described in [BHS92]. It basically consists on re-timestamping the hash of the document as well as the original timestamp before the hash function is broken.

We build two trees in parallel for each round using two different hash functions (SHA-1 and RIPEMD-160). In that way, the system remains secure in the case of an unexpected break of one of the hash functions used.

DESCRIPTION AND ANALYSIS OF THE TIMESEC TIMESTAMPING IMPLEMENTATION

We will now introduce the basic design of the system we have developed, which is based on the technique introduced above.

Initially, the user designates a document to be timestamped. Two hashes of it are created using the SHA-1 and RIPEMD-160 algorithms. The request

containing the two hashes is then sent by the client to the STA . Upon request receipt, the STA creates the corresponding timestamp using the following process.

Main description of the timestamping process

The system design follows a highly decoupled multi-threaded approach. Each step is assigned to a specific component, which has its own different thread. In the Figure 2 we present a schematic outline of the process. The multi-thread approach is justified by the requirement to obtain a highly responsive and load independent implementation. By isolating the process charges into independent steps we try to decouple the load between them. Each step has also a working queue. Those queues are in charge of softening the speed differences between the different process steps.

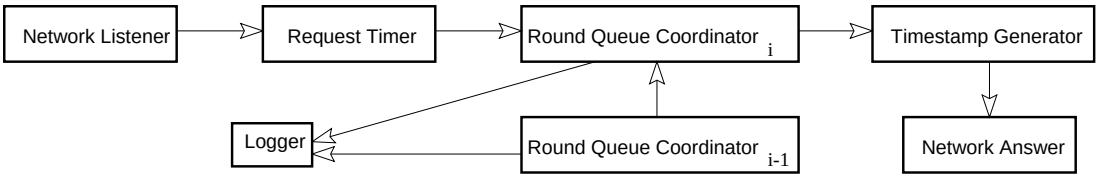


Figure 2: Interactions between the components

The "Network Listener" is in charge of continuously listen to the clients' timestamp requests. The "Request Timer" receives the constructed requests from the "Network Listener". Then, it times and forwards them to the actual "Round Queue Coordinator". Each round has its own "Round Queue Coordinator", which is in charge of compiling and processing into a tree all the requests belonging to the round. When the round tree has been computed it is forwarded to the "Timestamp Generator", which generates the corresponding timestamps. Once a timestamp is generated, the "Timestamp Generator" forwards it to the "Network Answer", which in turn forwards it to the client.

The Network Listener

The "Network Listener" responsibility is to listen the network continuously for timestamping requests. When it receives a data stream, the "Network Listener" checks it in order to determine if it is a valid request. In the case it is, it sends an affirmative contact response to the client, it creates a "Timestamp Request" object and adds it to the "Request Timer" queue. Then it goes back to listen

to the network. In the case the request message is not correct, it sends an error message to the client.

We tried to give as few tasks as possible to the "Network Listener" to let it listen the network, which is its primary task. In order to improve the overall performance, and to avoid the fact that a slow client connection could affect the other ones, several copies of the "Network Listener" can be active at the same time.

The Request Timer

There is only an instance of "The Request Timer" in the system. The "Request Timer" is in charge of ordering the requests received from the several "Network Listeners" and timing them accordingly. All delays introduced by the system before that point (namely, those introduced by the "Network Listener") are indistinguishable from network delays, and thus not taken into account. Once a request has been timed, the "Request Timer" tries to add it to the current round queue. As the rounds are closed asynchronously by the corresponding "Round Queue Coordinator" this operation is not always successful, in that case, the "Request Timer" re-times the request and retries to queue it until it finds an open round. In that process the request sequence is preserved in order to provide a consistent behavior.

Round Queue Coordinator creation: "Round Queue Coordinator" instances are created by the "Request Timer" upon processing a request corresponding to a non-existing round. The creation of the rounds that have no requests is delayed until a request is received. Once created, those empty rounds are immediately processed, introducing no significant delay into the process.

Round number determination: Round numbers form a non-interrupted increasing integer sequence. Rounds are always in synchronization with the round duration intervals. In other words, if the round duration is one minute, all rounds will start in an absolute minute boundary, independently from when the system has been started. "Big Rounds" are determined by the "Request Timer" using a similar approach to the one followed to determine the round boundaries. We do not restrict the duration of the round to a fixed value for the lifetime of the STA. To achieve this, the information about round and "Big Round" duration is introduced into the system at the start-up phase. If we wish to modify it, we must

first shutdown the system, change the values and then restart the system, which is the only safe procedure we had foreseen.

The Round Queue Coordinator

The first thing a "Round Queue Coordinator" does is to determine the offset between the actual time and the round due time. Requests will be accepted only if the round is still valid (round is open). When requested by the "Request Timer", the "Round Queue Coordinator" adds the request to the queue and logs it. This logged request will be latter used for process auditing purposes.

When the round time is over, it obtains the "Round Values" from the preceding round and it computes the round binary trees (one for each hash algorithm) to obtain the corresponding "Round Values". Then it gives the computed trees to the "Timestamp Generator" and finally adds to the log the "Round Values" and the "Round Root Values". Those logged values will be latter used for timestamp verification and process auditing purposes. If the actual round is a "Big Round" those values are forwarded to a fixed media as well.

As you may have noticed in the section "Introduction of the timestamping techniques", the binary tree is defined for a number of leafs (requests) that is a power of 2. In general, this is not the case. We could create fake requests to finish the tree, but this will add a lot of requests (if we have $2^n + 1$ requests, then we will need to add $2^n - 1$ fake requests). A smarter solution is to add a random value only when we need it. Then, we add at most n values (one for each level of the tree). We call these nodes "Special Node", which will be logged as well. Instead of random values we could choose to use 0 or another fixed value, this would be as secure as our choice if the hash functions were "perfect". As hash functions are only "presumably perfect", we though that we could made our design more secure with really few additional computations.

In our implementation, the STA queues the requests and computes the tree at the end of the round. At first sight, it could seem a more natural solution to build the tree as soon as the requests arrive. At the end of the round, the computation of the tree would then be ended by getting the last "Round Value" and computing the actual "Round Value". In fact, this solution is harder to implement, and has no effect on the security achieved as no one can check that the STA does not perform any reordering of the requests before it publishes the "Round Value".

The Timestamp Generator

The "Timestamp Generator" processes the round trees by pairs (one for each hash algorithm) in order to generate the timestamps for each of the requests contained in the trees. In order to maximize the system responsiveness, once a timestamp has been generated it is immediately forwarded to the "Network Answer". Finally, when all the timestamps contained in a round tree have been processed the tree is destroyed.

The Network Answer

The "Network Answer" is in charge of forwarding the processed timestamps to the clients. It has been specified in such a way that it can run several threads, in that way the rest of the timestamping process can be isolated from possible network delay problematic.

The timestamp verification process

First, the verifier designates a document and its corresponding timestamp for verification. Then, the verifier's system (his personal computer or a remote computer independent from the STA) generates the two document hashes and checks if they match with those contained in the timestamp. Afterwards, the "Round Value" is reconstructed using the data provided in the timestamp. If the computed "Round Value" is consistent with the one contained in the timestamp then the next step in the verification process is to compare this "Round Value" to the "Round Value" obtained from the STA repository. Finally, the verifier provides his system with the two "Big Round Values" that he founds in the "unmodifiable media"; the verifier's system gets all the necessary "Round Values" and "Root Round Values" from the STA and it checks the coherency of the two linking chains (one for each hash function).

The audit process

The auditor designates two "Big Rounds", which he fetches from a fixed media. The system behavior will be checked between these two "Big Round Values". For each round, the auditor's system gets all the hash values (leafs of the tree and "Special Nodes") and the "Round Value" from the STA. Then, it constructs the two trees and checks that the "Round Value" is consistent. These two steps are

repeated until all the considered rounds are checked or until an error has been found. In that way, all theoretically verifiable system behavior can be verified a posteriori.

The system start-up process

Here the most sensible issue is to be able to correctly start-up the system when an unexpected shutdown has occurred. If that is the case, the log will show an unfinished round; then the system marks all entries after the last complete round as invalid and publishes that round as a "Big Round". If the log was consistent, it accesses the last valid "Round Value" in the log and publishes it as a "Big Round". This process insures a fully verifiable behavior; we are able to detect non fully-processed requests.

The system shutdown process

The administrator signals the system to shutdown. No more timestamping requests are accepted. The system waits until the current round is finished and this "Round Value" is published as "Big Round".

REFERENCES

[BHS92] D. Bayer, S. Haber, and W.-S. Stornetta. Improving the efficiency and reliability of digital timestamping. In Springer Verlag, editor, *Sequences'91: Methods in Communication, Security, and Computer Science*, pages 329–334, 1992.

[HS91] S. Haber and W.-S. Stornetta. How to timestamp a digital document. *Journal of Cryptology*, 3(2):99–112, 1991.

[HS97] S. Haber and W.S. Stornetta. Secure names for bit-strings. In *Proceedings of the 4th ACM Conference on Computer and Communication Security*, pages 28–35. ACM Press, April 1997.

[MQ97] H. Massias and J.-J. Quisquater. Time and cryptography. Technical report, TIMESEC Project (Federal Governement Project, Belgium), 1997. Available at `http://www.dice.ucl.ac.be/crypto/TIMESEC.html`.

[PRQ+98] B. Preneel, B. Van Rompay, J.-J. Quiquater, H. Massias, and X. Serret Avila. Design of a timestamping system. Technical report, TIMESEC Project (Federal Governement Project, Belgium), 1998. To be available at `http://www.dice.ucl.ac.be/crypto/TIMESEC.html`.

J. Cryptology (1991) 3: 99–111

Journal of Cryptology

How To Time-Stamp a Digital Document[1]

Stuart Haber and W. Scott Stornetta
Bellcore, 445 South Street,
Morristown, NJ 07960-1910, U.S.A.
stuart@bellcore.com stornetta@bellcore.com

Abstract. The prospect of a world in which all text, audio, picture, and video documents are in digital form on easily modifiable media raises the issue of how to certify when a document was created or last changed. The problem is to time-stamp the data, not the medium. We propose computationally practical procedures for digital time-stamping of such documents so that it is infeasible for a user either to back-date or to forward-date his document, even with the collusion of a time-stamping service. Our procedures maintain complete privacy of the documents themselves, and require no record-keeping by the time-stamping service.

Key words. Time-stamp, Hash.

Time's glory is to calm contending kings,
To unmask falsehood, and bring truth to light,
To stamp the seal of time in aged things,
To wake the morn, and sentinel the night,
To wrong the wronger till he render right.

The Rape of Lucrece, l. 941

1. Introduction

In many situations there is a need to certify the date a document was created or last modified. For example, in intellectual property matters, it is sometimes crucial to verify the date an inventor first put in writing a patentable idea, in order to establish its precedence over competing claims.

One accepted procedure for time-stamping a scientific idea involves daily notations of one's work in a lab notebook. The dated entries are entered one after another in the notebook, with no pages left blank. The sequentially numbered, sewn-in pages of the notebook make it difficult to tamper with the record without leaving telltale signs. If the notebook is then stamped on a regular basis by a notary public or reviewed and signed by a company manager, the validity of the claim is further enhanced. If the precedence of the inventor's ideas is later challenged, both

[1] Date received: August 19, 1990. Date revised: October 26, 1990.

the physical evidence of the notebook and the established procedure serve to substantiate the inventor's claims of having had the ideas on or before a given date.

There are other methods of time-stamping. For example, one can mail a letter to oneself and leave it unopened. This ensures that the enclosed letter was created before the time postmarked on the envelope. Businesses incorporate more elaborate procedures into their regular order of business to enhance the credibility of their internal documents, should they be challenged at a later date. For example, these methods may ensure that the records are handled by more than one person, so that any tampering with a document by one person will be detected by another. But all these methods rest on two assumptions. First, the records can be examined for telltale signs of tampering. Second, there is another party that views the document whose integrity or impartiality is seen as vouchsafing the claim.

We believe these assumptions are called into serious question for the case of documents created and preserved exclusively in digital form. This is because electronic digital documents are so easy to tamper with, and the change need not leave any telltale sign on the physical medium. What is needed is a method of time-stamping digital documents with the following two properties. First, we must find a way to time-stamp the data itself, without any reliance on the characteristics of the medium on which the data appears, so that it is impossible to change even one bit of the document without the change being apparent. Second, it should be impossible to stamp a document with a time and data different from the actual one.

The purpose of this paper is to introduce a mathematically sound and computationally practical solution to the time-stamping problem. In the sections that follow, we first consider a naive solution to the problem, the digital safety-deposit box. This serves the pedagogical purpose of highlighting additional difficulties associated with digital time-stamping beyond those found in conventional methods of time-stamping. Successive improvements to this naive solution finally lead to practical ways to implement digital time-stamping.

2. The Setting

The setting for our problem is a distributed network of users, perhaps representing individuals, different companies, or divisions within a company; we refer to the users as *clients*. Each client has a unique identification number.

A solution to the time-stamping problem may have several parts. There is a procedure that is performed immediately when a client desires to have a document time-stamped. There should be a method for the client to verify that this procedure has been correctly performed. There should also be a procedure for meeting a third party's challenge to the validity of a document's time-stamp.

As with any cryptographic problem, it is a delicate matter to characterize precisely the security achieved by a time-stamping scheme. A good solution to the time-stamping problem is one for which, under reasonable assumptions about the computational abilities of the users of the scheme and about the complexity of a computational problem, and possibly about the trustworthiness of the users, it is difficult or impossible to produce false time-stamps. Naturally, the weaker the assumptions needed, the better.

3. A Naive Solution

A naive solution, a "digital safety-deposit box," could work as follows. Whenever a client has a document to be time-stamped, he or she transmits the document to a time-stamping service (TSS). The service records the date and time the document was received and retains a copy of the document for safe-keeping. If the integrity of the client's document is ever challenged, it can be compared with the copy stored by the TSS. If they are identical, this is evidence that the document has not been tampered with after the date contained in the TSS records. This procedure does in fact meet the central requirement for the time-stamping of a digital document.[2] However, this approach raises several concerns:

Privacy. This method compromises the privacy of the document in two ways: a third party could eavesdrop while the document is being transmitted, and after transmission it is available indefinitely to the TSS itself. Thus the client has to worry not only about the security of documents it keeps under its direct control, but also about the security of its documents at the TSS.

Bandwidth and storage. Both the amount of time required to send a document for time-stamping and the amount of storage required at the TSS depend on the length of the document to be time-stamped. Thus the time and expense required to time-stamp a large document might be prohibitive.

Incompetence. The TSS copy of the document could be corrupted in transmission to the TSS, it could be incorrectly time-stamped when it arrives at the TSS, or it could become corrupted or lost altogether at any time while it is stored at the TSS. Any of these occurrences would invalidate the client's time-stamping claim.

Trust. The fundamental problem remains: nothing in this scheme prevents the TSS from colluding with a client in order to claim to have time-stamped a document for a date and time different from the actual one.

In the next section we describe a solution that addresses the first three concerns listed above. The final issue, trust, is handled separately and at greater length in the following section.

4. A Trusted Time-Stamping Service

In this section we assume that the TSS is trusted, and describe two improvements on the naive solution above.

4.1. *Hash*

Our first simplification is to make use of a family of cryptographically secure *collision-free hash functions.* This is a family of functions $h: \{0, 1\}^* \to \{0, 1\}^l$ compressing bit-strings of arbitrary length to bit-strings of a fixed length l, with the following properties:

[2] The authors recently learned of a similar proposal sketched by Kanare [14].

1. The functions h are easy to compute, and it is easy to pick a member of the family at random.
2. It is computationally infeasible, given one of these functions h, to find a pair of distinct strings x, x' satisfying $h(x) = h(x')$. (Such a pair is called a *collision* for h.)

The practical importance of such functions has been known for some time, and researchers have used them in a number of schemes; see, for example, [7], [15], and [16]. Damgård gave the first formal definition, and a constructive proof of their existence, on the assumption that there exist one-way "claw-free" permutations [4]. For this, any "one-way group action" is sufficient [3].

Naor and Yung defined the similar notion of "universal one-way hash functions," which satisfy, in place of the second condition above, the slightly weaker requirement that it be computationally infeasible, given a string x, to compute another string $x' \neq x$ satisfying $h(x) = h(x')$ for a randomly chosen h. They were able to construct such functions on the assumption that there exist one-to-one one-way functions [17]. Rompel has recently shown that such functions exist if there exist one-way functions at all [20]. See Section 6.3 below for a discussion of the differences between these two sorts of cryptographic hash functions.

There are practical implementations of hash functions, for example, that of Rivest [19], which seem to be reasonably secure.

We use the hash functions as follows. Instead of transmitting his document x to the TSS, a client will send its hash value $h(x) = y$ instead. For the purposes of authentication, time-stamping y is equivalent to time-stamping x. This greatly reduces the bandwidth problem and the storage requirements, and solves the privacy issue as well. Depending on the design goals for an implementation of time-stamping, there may be a single hash function used by everybody, or different hash functions for different users.

For the rest of this paper, we speak of time-stamping hash values y—random-appearing bit-strings of a fixed length. Part of the procedure for validating a time-stamp will be to produce the preimage document x that satisfies $h(x) = y$; inability to produce such an x invalidates the putative time-stamp.

4.2. *Signature*

The second improvement makes use of digital signatures. Informally, a *signature scheme* is an algorithm for a party, the signer, to tag messages in a way that uniquely identifies the signer. Digital signatures were proposed by Rabin [18] and by Diffie and Hellman [7]. After a long sequence of papers by many authors, Rompel [20] showed that the existence of one-way functions can be used in order to design a signature scheme satisfying the very strong notion of security that was first defined by Goldwasser *et al.* [10].

With a secure signature scheme available, when the TSS receives the hash value, it appends the date and time, then signs this compound document and sends it to the client. By checking the signature, the client is assured that the TSS actually did process the request, that the hash was correctly received, and that the correct time is included. This takes care of the problem of present and future incompetence on the part of the TSS, and reduces the need for the TSS to store records.

5. Two Time-Stamping Schemes

Sed quis custodiet ipsos Custodes?
Juvenal, c. 100 A.D.
But who will guard the guards themselves?

What we have described so far is, we believe, a practical method for time-stamping digital documents of arbitrary length. However, neither the signature nor the use of hash functions in any way prevents a time-stamping service from issuing a false time-stamp. Ideally, we would like a mechanism which guarantees that no matter how unscrupulous the TSS is, the times it certifies will always be the correct ones, and that it will be unable to issue incorrect time-stamps even if it tries to.

It may seem difficult to specify a time-stamping procedure so as to make it impossible to produce fake time-stamps. After all, if the output of an algorithm A, given as input a document x and some timing information τ, is a bit-string $c = A(x, \tau)$ that stands as a legitimate time-stamp for x, what is to prevent a forger some time later from computing the same timing information τ and then running A to produce the same certificate c? The question is relevant even if A is a probabilistic algorithm.

Our task may be seen as the problem of simulating the action of a trusted TSS, in the absence of generally trusted parties. There are two rather different approaches we might take, and each one leads to a solution. The first approach is to constrain a centralized but possibly untrustworthy TSS to produce genuine time-stamps in such a way that fake ones are difficult to produce. The second approach is somehow to distribute the required trust among the users of the service. It is not clear that either of these can be done at all.

5.1. *Linking*

Our first solution begins by observing that the sequence of clients requesting time-stamps and the hashes they submit cannot be known in advance. So if we include bits from the previous sequence of client requests in the signed certificate, then we know that the time-stamp occurred after these requests. But the requirement of including bits from previous documents in the certificate can also be used to solve the problem of constraining the time in the other direction, because the time-stamping company cannot issue later certificates unless it has the current request in hand.

We describe two variants of this linking scheme; the first one, slightly simpler, highlights our main idea, while the second one may be preferable in practice. In both variants, the TSS makes use of a collision-free hash function, denoted H. This is in addition to clients' use of hash functions in order to produce the hash value of any documents that they wish to have time-stamped.

To be specific, a time-stamping *request* consists of an l-bit string y (presumably the hash value of the document) and a client identification number ID. We use $\sigma(\cdot)$ to denote the signing procedure used by the TSS. The TSS issues signed, sequentially numbered time-stamp *certificates*. In response to the request (y_n, ID_n) from our client, the nth request in sequence, the TSS does two things:

1. The TSS sends our client the signed certificate $s = \sigma(C_n)$, where the certificate

$$C_n = (n, t_n, \mathrm{ID}_n, y_n; L_n)$$

consists of the sequence number n, the time t_n, the client number ID_n and the hash value y_n from the request, and certain *linking information*, which comes from the previously issued certificate: $L_n = (t_{n-1}, \mathrm{ID}_{n-1}, y_{n-1}, H(L_{n-1}))$.
2. When the next request has been processed, the TSS sends our client the identification number ID_{n+1} for that next request.

Having received s and ID_{n+1} from the TSS, she checks that s is a valid signature of a good certificate, i.e., one that is of the correct form $(n, t, \mathrm{ID}_n, y_n; L_n)$, containing the correct time t.

If her time-stamped document x is later challenged, the challenger first checks that the time-stamp (s, ID_{n+1}) is of the correct form (with s being a signature of a certificate that indeed contains a hash of x). In order to make sure that our client has not colluded with the TSS, the challenger can call client ID_{n+1} and ask him to produce his time-stamp (s', ID_{n+2}). This includes a signature

$$s' = \sigma(n + 1, t_{n+1}, \mathrm{ID}_{n+1}, y_{n+1}; L_{n+1})$$

of a certificate that contains in its linking information L_{n+1} a copy of her hash value y_n. This linking information is further authenticated by the inclusion of the image $H(L_n)$ of her linking information L_n. An especially suspicious challenger now can call up client ID_{n+2} and verify the next time-stamp in the sequence; this can continue for as long as the challenger wishes. Similarly, the challenger can also follow the chain of time-stamps backward, beginning with client ID_{n-1}.

Why does this constrain the TSS from producing bad time-stamps? First, observe that the use of the signature has the effect that the *only* way to fake a time-stamp is with the collaboration of the TSS. But the TSS cannot forward-date a document, because the certificate must contain bits from requests that immediately preceded the desired time, yet the TSS has not received them. The TSS cannot feasibly back-date a document by preparing a fake time-stamp for an earlier time, because bits from the document in question must be embedded in certificates immediately following that earlier time, yet these certificates have already been issued. Furthermore, correctly embedding a new document into the already-existing stream of time-stamp certificates requires the computation of a collision for the hash function H.

Thus the only possible spoof is to prepare a fake chain of time-stamps, long enough to exhaust the most suspicious challenger that one anticipates.

In the scheme just outlined, clients must keep all their certificates. In order to relax this requirement, in the second variant of this scheme we link each request not just to the next request but to the next k requests. The TSS responds to the nth request as follows:

1. As above, the certificate C_n is of the form $C_n = (n, t_n, \mathrm{ID}_n, y_n; L_n)$, where now the linking information L_n is of the form

$$L_n = [(t_{n-k}, \mathrm{ID}_{n-k}, y_{n-k}, H(L_{n-k})), \ldots, (t_{n-1}, \mathrm{ID}_{n-1}, y_{n-1}, H(L_{n-1}))].$$

2. After the next k requests have been processed, the TSS sends our client the list $(\mathrm{ID}_{n+1}, \ldots, \mathrm{ID}_{n+k})$.

After checking that this client's time-stamp is of the correct form, a suspicious challenger can ask any one of the next k clients ID_{n+i} to produce his time-stamp. As above, his time-stamp includes a signature of a certificate that contains in its linking information L_{n+i} a copy of the relevant part of the challenged time-stamp certificate C_n, authenticated by the inclusion of the hash by H of the challenged client's linking information L_n. His time-stamp also includes client numbers $(\mathrm{ID}_{n+i+1}, \ldots, \mathrm{ID}_{n+i+k})$, of which the last i are new ones; the challenger can ask these clients for their time-stamps, and this can continue for as long as the challenger wishes.

In addition to easing the requirement that clients save all their certificates, this second variant also has the property that correctly embedding a new document into the already-existing stream of time-stamp certificates requires the computation of a simultaneously k-wise collision for the hash function H, instead of just a pairwise collision.

5.2. *Distributed Trust*

For this scheme we assume that there is a secure signature scheme so that each user can sign messages, and that a standard secure pseudorandom generator G is available to all users. A *pseudorandom generator* is an algorithm that stretches short input *seeds* to output sequences that are indistinguishable by any feasible algorithm from random sequences; in particular, they are unpredictable. Such generators were first studied by Blum and Micali [2] and by Yao [22]; Impagliazzo *et al.* have shown that they exist if there exist one-way functions [12].

Once again, we consider a hash value y that our client would like to time-stamp. She uses y as a seed for the pseudorandom generator, whose output can be interpreted in a standard way as a k-tuple of client identification numbers:

$$G(y) = (\mathrm{ID}_1, \mathrm{ID}_2, \ldots, \mathrm{ID}_k).$$

Our client sends her request (y, ID) to each of these clients. She receives in return from client ID_j a signed message $s_j = \sigma_j(t, \mathrm{ID}, y)$ that includes the time t. Her time-stamp consists of $[(y, \mathrm{ID}), (s_1, \ldots, s_k)]$. The k signatures s_j can easily be checked by our client or by a would-be challenger. No further communication is required in order to meet a later challenge.

Why should such a list of signatures constitute a believable time-stamp? The reason is that in these circumstances, the only way to produce a time-stamped document with an incorrect time is to use a hash value y so that $G(y)$ names k clients that are willing to cooperate in faking the time-stamp. If at any time there is at most a constant fraction ε of possibly dishonest clients, the expected number of seeds y that have to be tried before finding a k-tuple $G(y)$ containing only collaborators from among this fraction is ε^{-k}. Furthermore, since we have assumed that G is a secure pseudorandom generator, there is no faster way of finding such a convenient seed y than by choosing it at random. This ignores the adversary's further problem,

in most real-world scenarios, of finding a plausible document that hashes to a convenient value y.

The parameter k should be chosen when designing the system so that this is an infeasible computation. Observe that even a highly pessimistic estimate of the percentage of the client population that is corruptible—ε could be 90%—does not entail a prohibitively large choice of k. In addition, the list of corruptible clients need not be fixed, as long as their fraction of the population never exceeds ε.

This scheme need not use a centralized TSS at all. The only requirements are that it be possible to call up other clients at will and receive from them the required signatures, and that there be a public directory of clients so that it is possible to interpret the output of $G(y)$ in a standard way as a k-tuple of clients. A practical implementation of this method would require provisions in the protocol for clients that cannot be contacted at the time of the time-stamping request. For example, for suitable $k' < k$, the system might accept signed responses from any k' of the k clients named by $G(y)$ as a valid time-stamp for y (in which case a greater value for the parameter k would be needed in order to achieve the same low probability of finding a set of collaborators at random).

6. Remarks

6.1. *Tradeoffs*

There are a number of tradeoffs between the two schemes. The distributed-trust scheme has the advantage that all processing takes place when the request is made. In the linking scheme, on the other hand, the client has a short delay while she waits for the second part of her certificate; and meeting a later challenge may require further communication.

A related disadvantage of the linking scheme is that it depends on at least some parties (clients or, perhaps, the TSS) storing their certificates.

The distributed-trust scheme makes a greater technological demand on the system: the ability to call up and demand a quick signed response at will.

The linking scheme only locates the time of a document between the times of the previous and the next requests, so it is best suited to a setting in which relatively many documents are submitted for time-stamping, compared with the scale at which the timing matters.

It is worth remarking that the time-constraining properties of the linking scheme do not depend on the use of digital signatures.

6.2. *Time Constraints*

We would like to point out that our schemes constrain the event of time-stamping both forward and backward in time. However, if any amount of time may pass between the creation of a document and when it is time-stamped, then no method can do more than forward-constrain the time at which the document itself was created. Thus, in general, time-stamping should only be considered as evidence that a document has not been back-dated.

On the other hand, if the time-stamping event can be made part of the document creation event, then the constraint holds in both directions. For example, consider the sequence of phone conversations that pass through a given switch. In order to process the next call on this switch, we could require that linking information be provided from the previous call. Similarly, at the end of the call, linking information would be passed onto the next call. In this way, the document creation event (the phone call) includes a time-stamping event, and so the time of the phone call can be fixed in both directions. The same idea could apply to sequential financial transactions, such as stock trades or currency exchanges, or any sequence of electronic interactions that take place over a given physical connection.

6.3. *Theoretical Considerations*

Although we do not do it here, we suggest that a precise complexity-theoretic definition of the strongest possible level of time-stamping security could be given along the lines of the definitions given by Goldwasser and Micali [9], Goldwasser *et al.* [10], and Galil *et al.* [8] for various cryptographic tasks. The time-stamping and the verification procedures would all depend on a *security parameter p*. A time-stamp scheme would be *polynomially secure* if the success probability of a polynomially bounded adversary who tries to manufacture a bogus time-stamp is smaller than any given polynomial in $1/p$ for sufficiently large p.

Under the assumption that there exist one-way claw-free permutations, we can prove our linking scheme to be polynomially secure. If we assume that there is always at most a constant fraction of corruptible clients, and assuming as well the existence of one-way functions (and therefore the existence of pseudorandom generators and of a secure signature scheme), we can prove our distributed-trust scheme to be polynomially secure.

In Section 4.1 above we mentioned the difference between "collision-free" and "universal one-way" hash functions. The existence of one-way functions is sufficient to give us universal one-way hash functions. However, in order to prove the security of our time-stamping schemes, we apparently need the stronger guarantee of the difficulty of producing hash collisions that is provided by the definition of collision-free hash functions. As far as is currently known, a stronger complexity assumption—namely, the existence of claw-free pairs of permutations—is needed in order to prove the existence of these functions. (See also [5] and [6] for further discussion of the theoretical properties of cryptographic hash functions.)

Universal one-way hash functions were the tool used in order to construct a secure signature scheme. Our apparent need for a stronger assumption suggests a difference, perhaps an essential one, between signatures and time-stamps. It is in the signer's own interest to act correctly in following the instructions of a secure signature scheme (for example, in choosing a hash function at random from a certain set). For time-stamping, on the other hand, a dishonest user or a colluding TSS may find it convenient not to follow the standard instructions (for example, by choosing a hash function so that collisions are easy to find); the time-stamping scheme must be devised so that there is nothing to be gained from such misbehavior.

If it is possible, we would like to reduce the assumptions we require for secure

time-stamping to the simple assumption that one-way functions exist. This is the minimum reasonable assumption for us, since all of complexity-based cryptography requires the existence of one-way functions [12], [13].

6.4. *Practical Considerations*

As we move from the realm of complexity theory to that of practical cryptosystems, new questions arise. In one sense, time-stamping places a heavier demand on presumably one-way functions than would some other applications. For example, if an electronic funds transfer system relies on a one-way function for authentication, and that function is broken, then all of the transfers carried out before it was broken are still valid. For time-stamps, however, if the hash function is broken, then all of the time-stamps issued prior to that time are called into question.

A partial answer to this problem is provided by the observation that time-stamps can be renewed. Suppose we have two time-stamping implementations, and that there is reason to believe that the first implementation will soon be broken. Then certificates issued using the old implementation can be renewed using the new implementation. Consider a time-stamp certificate created using the old implementation that is time-stamped with the new implementation before the old one is broken. Prior to the old implementation's breaking, the only way to create a certificate was by legitimate means. Thus, by time-stamping the certificate itself with the new implementation, we have evidence not only that the document existed prior to the time of the new time-stamp, but that it existed at the time stated in the original certificate.

Another issue to consider is that producing hash collisions alone is not sufficient to break the time-stamping scheme. Rather, meaningful documents must be found which lead to collisions. Thus, by specifying the format of a document class, we can complicate the task of finding meaningful collisions. For example, the density of ASCII-only texts among all possible bit-strings of length N bytes is $(2^7/2^8)^N$, or $1/2^N$, simply because the high-order bit of each byte is always 0. Even worse, the density of acceptable English text can be bounded above by an estimate of the entropy of English as judged by native speakers [21]. This value is approximately 1 bit per ASCII character, giving a density of $(2^1/2^8)^N$, or $1/128^N$.

We leave it to future work to determine whether we can formalize the increased difficulty of computing collisions if valid documents are sparsely and perhaps randomly distributed in the input space. Similarly, the fact that a k-way linking scheme requires the would-be adversary to compute k-way collisions rather than collision pairs may be parlayed into relaxing the requirements for the hash function. It may also be worthwhile to explore when there exist hash functions for which there are *no* k-way collisions among strings in a suitably restricted subset of the input space; the security of such a system would no longer depend on a complexity assumption.

7. Applications

Using the theoretically best (cryptographically secure) hash functions, signature schemes, and pseudorandom generators, we have designed time-stamping schemes

that possess theoretically desirable properties. However, we would like to emphasize the practical nature of our suggestion: because there are *practical* implementations of these cryptographic tools, both of our time-stamp schemes can be inexpensively implemented as described. Practical hash functions like Rivest's are quite fast, even running on low-end PCs [19].

What kinds of documents would benefit from secure digital time-stamping? For documents that establish the precedence of an invention or idea, time-stamping has a clear value. A particularly desirable feature of digital time-stamping is that it makes it possible to establish precedence of intellectual property without disclosing its contents. This could have a significant effect on copyright and patent law, and could be applied to everything from software to the secret formula for Coca-Cola.

But what about documents where the date is not as significant as simply whether or not the document has been tampered with? These documents can benefit from time-stamping, too, under the following circumstances. Suppose we can establish that either the necessary knowledge or the motivation to tamper with a document did not exist until long after the document's creation. For example, we can imagine a company that deals with large numbers of documents each day, some few of which are later found to be incriminating. If all the company's documents were routinely time-stamped at the time of their creation, then by the time it became apparent which documents were incriminating and how they needed to be modified, it would be too late to tamper with them. We call such documents *tamper-unpredictable.* It seems clear that many business documents are tamper-unpredictable. Thus, if time-stamping were to be incorporated into the established order of business, the credibility of many documents could be enhanced.

A variation that may be particularly useful for business documents is to time-stamp a log of documents rather than each document individually. For example, each corporate document created in a day could be hashed, and the hash value added to the company's daily log of documents. Then, at the end of the business day, the log alone could be submitted for time-stamping. This would eliminate the expense of time-stamping each document individually, while still making it possible to detect tampering with each document; we could also determine whether any documents had been destroyed altogether.

Of course, digital time-stamping is not limited to text documents. Any string of bits can be time-stamped, including digital audio recordings, photographs, and full-motion videos. Most of these documents are tamper-unpredictable. Therefore, time-stamping can help to distinguish an original photograph from a retouched one, a problem that has received considerable attention of late in the popular press [1], [11]. It is in fact difficult to think of any other algorithmic "fix" that could add more credibility to photographs, videos, or audio recordings than time-stamping.

8. Summary

In this paper we have shown that the growing use of text, audio, and video documents in digital form and the ease with which such documents can be modified creates a new problem: how can we certify when a document was created or last modified? Methods of certification, or time-stamping, must satisfy two criteria.

First, they must time-stamp the actual bits of the document, making no assumptions about the physical medium on which the document is recorded. Second, the date and time of the time-stamp must not be forgeable.

We have proposed two solutions to this problem. Both involve the use of one-way hash functions, whose outputs are processed in lieu of the actual documents, and of digital signatures. The solutions differ only in the way that the date and time are made unforgeable. In the first, the hashes of documents submitted to a TSS are linked together, and certificates recording the linking of a given document are distributed to other clients both upstream and downstream from that document. In the second solution, several members of the client pool must time-stamp the hash. The members are chosen by means of a pseudorandom generator that uses the hash of the document itself as seed. This makes it infeasible to choose deliberately which clients should and should not time-stamp a given hash. The second method could be implemented without the need for a centralized TSS at all.

Finally, we have considered whether time-stamping could be extended to enhance the authenticity of documents for which the time of creation itself is not the critical issue. This is the case for a large class of documents which we call "tamper-unpredictable." We further conjecture that no purely algorithmic scheme can add any more credibility to a document than time-stamping provides.

Acknowledgments

We gratefully acknowledge helpful discussions with Donald Beaver, Shimon Even, George Furnas, Burt Kaliski, Ralph Merkle, Jeff Shrager, Peter Winkler, Yacov Yacobi, and Moti Yung.

References

[1] J. Alter. When photographs lie. *Newsweek*, pp. 44–45, July 30, 1990.

[2] M. Blum and S. Micali. How to generate cryptographically strong sequences of pseudo-random bits. *SIAM J. Comput.*, **13**(4): 850–864, Nov. 1984.

[3] G. Brassard and M. Yung. One-way group actions. In *Advances in Cryptology—Crypto* '90. Lecture Notes in Computer Science, Springer-Verlag, Berlin, to appear.

[4] I. Damgård. Collision-free hash functions and public-key signature schemes. In *Advances in Cryptology—Eurocrypt* '87, pp. 203–217. Lecture Notes in Computer Science, vol. 304, Springer-Verlag, Berlin, 1988.

[5] I. Damgård. A design principle for hash functions. In *Advances in Cryptology—Crypto* '89 (ed. G. Brassard), pp. 416–427. Lecture Notes in Computer Science, vol. 435, Springer-Verlag, Berlin, 1990.

[6] A. DeSantis and M. Yung. On the design of provably secure cryptographic hash functions. In *Advances in Cryptology—Eurocrypt* '90. Lecture Notes in Computer Science, Springer-Verlag, Berlin, to appear.

[7] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Trans. Inform. Theory*, **22**: 644–654, Nov. 1976.

[8] Z. Galil, S. Haber, and M. Yung. Interactive public-key cryptosystems. Submitted for publication, 1990.

[9] S. Goldwasser and S. Micali. Probabilistic encryption. *J. Comput. System Sci.*, **28**: 270–299, April 1984.

[10] S. Goldwasser, S. Micali, and R. Rivest. A secure digital signature scheme. *SIAM J. Comput.*, **17**(2): 281–308, 1988.

[11] A. Grundberg. Ask it no questions: The camera can lie. *The New York Times*, Section 2, pp. 1, 29, August 12, 1990.

[12] R. Impagliazzo, L. Levin, and M. Luby. Pseudorandom generation from one-way functions. In *Proc.* 21st *STOC*, pp. 12–24. ACM, New York, 1989.

[13] R. Impagliazzo and M. Luby. One-way functions are essential for complexity-based cryptography. In *Proc.* 30th *FOCS*, pp. 230–235. IEEE, New York, 1989.

[14] H. M. Kanare. *Writing the Laboratory Notebook*, p. 117. American Chemical Society, Washington, D.C., 1985.

[15] R. C. Merkle. Secrecy, authentication, and public-key systems. Ph.D. thesis, Stanford University, 1979.

[16] R. C. Merkle. One-way hash functions and DES. In *Advances in Cryptology—Crypto* '89 (ed. G. Brassard), pp. 428–446. Lecture Notes in Computer Science, vol. 435, Springer-Verlag, Berlin, 1990.

[17] M. Naor and M. Yung. Universal one-way hash functions and their cryptographic applications. In *Proc.* 21st *STOC*, pp. 33–43. ACM, New York, 1989.

[18] M. O. Rabin. Digitalized signatures. In *Foundations of Secure Computation* (ed. R. A. DeMillo *et al.*), pp. 155–168. Academic Press, New York, 1978.

[19] R. Rivest. The MD4 message digest algorithm. In *Advances in Cryptology—Crypto* '90. Lecture Notes in Computer Science, Springer-Verlag, Berlin, to appear.

[20] J. Rompel. One-way functions are necessary and sufficient for secure signatures. In *Proc.* 22nd *STOC*, pp. 387–394. ACM, New York, 1990.

[21] C. Shannon. Prediction and entropy of printed English. *Bell System Tech. J.*, **30**: 50–64, 1951.

[22] A. C. Yao. Theory and applications of trapdoor functions. In *Proc.* 23rd *FOCS*, pp. 80–91. IEEE, New York, 1982.

Improving the Efficiency and Reliability of Digital Time-Stamping

Dave Bayer*
Barnard College
Columbia University
New York, N.Y. 10027 U.S.A.
dab@math.columbia.edu

Stuart Haber
Bellcore
445 South Street
Morristown, N.J. 07960 U.S.A.
stuart@bellcore.com

W. Scott Stornetta
Bellcore
445 South Street
Morristown, N.J. 07960 U.S.A.
stornetta@bellcore.com

March 1992

Abstract

To establish that a document was created after a given moment in time, it is necessary to report events that could not have been predicted before they happened. To establish that a document was created before a given moment in time, it is necessary to cause an event based on the document, which can be observed by others. Cryptographic hash functions can be used both to report events succinctly, and to cause events based on documents without revealing their contents. Haber and Stornetta have proposed two schemes for digital time-stamping which rely on these principles [HaSt 91].

We reexamine one of those protocols, addressing the resource constraint required for storage and verification of time-stamp certificates. By using trees, we show how to achieve an exponential increase in the publicity obtained for each time-stamping event, while reducing the storage and the computation required in order to validate a given certificate.

We show how time-stamping can be used in certain circumstances to extend the useful lifetime of different kinds of cryptographic certifications of authenticity, in the event that the certifying protocol is compromised. This can be applied to digital signatures, or to time-stamping itself, making the digital time-stamping process renewable.

*Partially supported by NSF grant DMS-90-06116.

1 Introduction

Causality fixes events in time. If an event was determined by certain earlier events, and determines certain subsequent events, then the event is sandwiched securely into its place in history. Fundamentally, this is why paper documents have forensic qualities allowing them to be accurately dated and examined for signs of after-the-fact tampering. However, documents kept in digital form need not be closely tied to any physical medium, and tampering may not leave any tell-tale signs in the medium.

Could an analogous notion of causality be applied to digital documents to correctly date them, and to make undetected tampering infeasible? Any solution would have to time-stamp the data itself, without any reliance on the properties of a physical medium, and would be especially useful and trustworthy if the date and time of the time-stamp could not be forged.

In [HaSt 91], Haber and Stornetta posed this problem, and proposed two solutions. Both involve the use of cryptographic hash functions (discussed in §2 below), whose outputs are processed in lieu of the actual documents. In the *linking* solution, the hash values of documents submitted to a time-stamping service are chained together in a linear list into which nothing can feasibly be inserted or substituted and from which nothing can feasibly be deleted. This latter property is insured by a further use of cryptographic hashing. In the *random-witness* solution, several members of the client pool must date and sign the hash value; their signatures form a composite certification that the time-stamp request was witnessed. These members are chosen by means of a pseudorandom generator that uses the hash of the document itself as a seed. This makes it infeasible to deliberately choose which clients should and should not act as witnesses.

In both of these solutions, the record-keeping requirements per time-stamping request are proportional to the number of (implicit) observers of the event. In §3 below we address the following problem: What if an immense flood of banal transactions want their time-stamps to become part of the historical record, but history just isn't interested? We propose to merge many unnoteworthy time-stamping events into one noteworthy event, using a tournament run by its participants. The winner can be easily and widely publicized. Each player, by remembering a short list of opponents, can establish participation in the tournament. We do this by building trees in place of the linked list of the linking solution, thus achieving an exponential increase in the number of observers. Such hash trees were previously used by Merkle [Merk 80] for a different purpose, to produce authentication certificates for a directory of public enciphering keys.

There are several ways in which a cryptographic system can be compromised. For example, users' private keys may be revealed; imprudent choice of key-lengths may be overtaken by an increase in computing power; and improved algorithmic techniques may render feasible the heretofore intractable computational problem on which the system is based. In §4 below we show how time-stamping can be used in certain circumstances to extend the useful lifetime of digital signatures. Applying the same technique to time-stamping itself, we demonstrate that digital time-stamps can be

renewed.

Finally, in §5 we discuss the relationships between the different methods of digital time-stamping that have been proposed.

2 Hash functions

The principal tool we use in specifying digital time-stamping schemes, here as in [HaSt 91], is the idea of a cryptographic hash function. This is a function compressing digital documents of arbitrary length to bit-strings of a fixed length, for which it is computationally infeasible to find two different documents that are mapped by the function to the same *hash value*. (Such a pair is called a *collision* for the hash function.) Hence it is infeasible to fabricate a document with a given hash value. In particular, a fragment of a document cannot be extended to a complete document with a given hash value, unless the fragment was known before the hash value was created. In brief, a hash value must follow its associated document in time.

There are practical implementations of hash functions, for example those of Rivest [Riv 90] and of Brachtl, *et al.* [BC$^+$ 88], which seem to be reasonably secure.

In a more theoretical vein, Damgard defined a family of *collision-free hash functions* to be a family of functions $h : \{0,1\}^* \to \{0,1\}^l$ compressing bit-strings of arbitrary length to bit-strings of a fixed length l, with the following properties:

1. The functions h are easy to compute, and it is easy to pick a member of the family at random.

2. It is computationally infeasible, given a random choice of one of these functions h, to find a pair of distinct strings x, x' satisfying $h(x) = h(x')$.

He gave a constructive proof of their existence, on the assumption that there exist one-way "claw-free" permutations [Dam 87]. For further discussion of theoretical questions relating to the existence of families of cryptographic hash functions (variously defined) see [HaSt 91] and the references contained therein.

In the rest of this paper, we will assume that a cryptographic hash function h is given: either a particular practical implementation, or one that has been chosen at random from a collision-free family.

3 Trees

In the linking scheme, the challenger of a time-stamp is satisfied by following the linked chain from the document in question to a time-stamp certificate that the challenger considers trustworthy. If a trustworthy certificate occurs about every N documents, say, then the verification process may require as many as N steps. We may reduce this cost from N to $\log N$, as follows.

Suppose we combine the hash values of two users' documents into one new hash value, and publicize only the combined hash value. (We will consider a "publicized"

value to be trustworthy.) Either participant, by saving his or her own document as well as the other contributing hash value, can later establish that the document existed before the time when the combined hash value was publicized.

More generally, suppose that N hash values are combined into one via a binary tree, and the resulting single hash value is widely publicized. To later establish priority, a participant need only record his own document, as well as the $\lceil \log_2 N \rceil$ hash values that were directly combined with the document's hash value along the path to the root of the tree. In addition, along with each combining hash value, the user needs to record its "handedness," indicating whether the newly computed value was placed before or after the combining hash value. Verification consists simply of recomputing the root of the tree from this data.

Once hashing functions are chosen, such a scheme could be carried out like a world championship tournament: Heterogeneous local networks could govern local subtrees under the scrutiny of local participants, and regional "winners" could be combined into global winners under the scrutiny of all interested parties. Global communication facilities are required, and a broadcast protocol must be agreed upon, but no centralized service bureau need administer or profit from this system. For example, given any protocol acceptable separately to the western and eastern hemispheres for establishing winners for a given one-hour time period, the winners can be broadcast by various members of the respective hemispheres, and anyone who wants to can carry out the computations to determine the unique global winner for that time period. Winners for shorter time periods can similarly be combined into unique winners for longer time periods, by any interested party.

At a minimum, daily global winners could be recorded in newspaper advertisements, to end up indefinitely on library microfilm. The newspaper functions as a widely available public record whose long-term preservation at many locations makes tampering very difficult. An individual who retains the set of values tracing the path between his document and the hash value appearing in the newspaper could establish the time of his document, without any reliance on other records. Anyone who wishes to be able to resolve time-stamps to greater accuracy needs only to record time-stamp broadcasts to greater accuracy.

4 Using time-stamping to extend the lifetime of a threatened cryptographic operation

The valid lifetime of a digitally signed document can be extended with digital time-stamping, in the following way. Imagine an implementation of a particular digital signature scheme, with a particular choice of key lengths, and consider a plaintext document D and its digital signature σ by a particular user. Now let the pair (D, σ) be time-stamped. Some time later the signature may become invalid, for any of a variety of reasons, including the compromise of the user's private key, an increase in available computing power making signatures with keys of that length unsafe, or the discovery of a basic flaw in the signature scheme. At that point, the document-

signature pair becomes questionable, because it may be possible for someone other than the original signer to create valid signatures.

However, if the pair (D, σ) was time-stamped at a time before the signature was compromised, then the pair still constitutes a valid signature. This is because it is known to have been created at a time when only legitimate users could have produced it. Its validity is not in question even though new signatures generated by the compromised method might no longer be trustworthy.

The same technique applies to other instances of cryptographic protocols. In particular, the technique can be used to renew the time-stamping process itself. Once again, imagine an implementation of a particular time-stamping scheme, and consider the pair (D, C), where C is a valid time-stamp certificate (in this implementation) for the document D. If (D, C) is time-stamped by an improved time-stamping method before the original method is compromised, then one has evidence not only that the document existed prior to the time of the new time-stamp, but that it existed at the time stated in the original certificate. Prior to the compromise of the old implementation, the only way to create a certificate was by legitimate means. (The ability to renew time-stamps was mentioned in [HaSt 91] but an incorrect method was given. The mistake of the previous work was in assuming that it is sufficient to renew the certificate alone, and not the document-certificate pair. This fails, of course, if the compromise in question is a method of computing hash collisions for the hash function used in submitting time-stamp requests.)

5 Different methods of time-stamping

To date, three different digital time-stamping techniques have been proposed: linear linking, random witness and linking into trees. What is the relationship between them? Does one supersede the others? Initially, one might think that trees satisfy time-stamping requirements better than the two previously proposed methods, because the tree protocol seems to reduce storage requirements while increasing the number of interested parties who serve as witnesses. But there are other tradeoffs to consider.

First we consider the linking protocol. In certain applications, such as a laboratory notebook, it is crucial not only to have a trustworthy date for each entry but also to establish in a trustworthy manner the exact sequence in which all entries were made. Linear linking of one entry to the next provides the most straightforward means of achieving this.

Next we consider the difference between the random-witness method and the tree method. While trees increase the number of witnesses to a given time-stamping event in proportion to the number of documents time-stamped, they do not guarantee a minimum number of witnesses. Neither do they guarantee that witnesses will retain their records. In contrast, in random witness the effective number of witnesses is the entire population, though only a small fraction are actually involved in any given time-stamping event. Furthermore, the set of signatures computed by the random-witness protocol explicitly creates a certificate which is evidence that a time-stamping

event was widely witnessed. Thus, the protocol does not depend for its final validity on witnesses keeping records. Random witness is somewhat analogous to placing an advertisement in the newspaper, as discussed earlier, but with an additional refinement. Like the newspaper ad, it is effectively a widely witnessed event, but in addition it creates a record of the witnessing.

Given these tradeoffs, we imagine that the three methods may be used in a complementary fashion, as the following example illustrates. An individual or company might use linear linking to time-stamp its own accounting records, sending the final summary value for a given time period to a service maintained by a group of individuals or parties. This service constructs linked trees at regular intervals. The root of each tree is then certified as a widely viewed event by using the random-witness protocol among the participants. In this way, individual and group storage needs can be minimized, and the number of events which require an official record of witnessing can be greatly reduced.

References

[BC$^+$ 88] B. O. Brachtl, D. Coppersmith, M. M. Hyden, S. M. Matyas, Jr., C. H. W. Meyer, J. Oseas, Sh. Pilpel, and M. Shilling. Data authentication using modification detection codes based on a public one way encryption function. U.S. Patent No. 4,908,861, issued March 13, 1990. (*Cf.* C. H. Meyer and M. Shilling, Secure program load with modification detection code. In *Securicom 88: 6ème Congrès mondial de la protection et de la sécurité informatique et des communications*, pp. 111–130 (Paris, 1988).)

[Dam 87] I. Damgård. Collision-free hash functions and public-key signature schemes. In *Advances in Cryptology—Eurocrypt '87*, Lecture Notes in Computer Science, Vol. 304, pp. 203–217, Springer-Verlag (Berlin, 1988).

[HaSt 91] S. Haber, W. S. Stornetta, How to time-stamp a digital document, *Journal of Cryptography*, Vol. 3, No. 2, pp. 99–111 (1991). (Presented at Crypto '90.)

[Merk 80] R. C. Merkle, Protocols for public key cryptosystems. In *Proc. 1980 Symp. on Security and Privacy*, IEEE Computer Society, pp. 122–133 (Apr. 1980).

[Riv 90] R. L. Rivest. The MD4 message digest algorithm. In *Advances in Cryptology—Crypto '90*, Lecture Notes in Computer Science, Vol. 537 (ed. A. J. Menezes, S. A. Vanstone), pp. 303–311, Springer-Verlag (Berlin, 1991).

Hashcash - A Denial of Service Counter-Measure

Adam Back
e-mail: adam@cypherspace.org

1st August 2002

Abstract

Hashcash was originally proposed as a mechanism to throttle systematic abuse of un-metered internet resources such as email, and anonymous remailers in May 1997. Five years on, this paper captures in one place the various applications, improvements suggested and related subsequent publications, and describes initial experience from experiments using hashcash.

The *hashcash* CPU cost-function computes a token which can be used as a proof-of-work. Interactive and non-interactive variants of cost-functions can be constructed which can be used in situations where the server can issue a challenge (connection oriented interactive protocol), and where it can not (where the communication is store–and–forward, or packet oriented) respectively.

Key Words: hashcash, cost-functions

1 Introduction

Hashcash [1] was originally proposed as a mechanism to throttle systematic abuse of un-metered internet resources such as email, and anonymous remailers in May 1997. Five years on, this paper captures in one place the various applications, improvements suggested and related subsequent publications, and describes initial experience from experiments using hashcash.

The *hashcash* CPU cost-function computes a token which can be used as a proof-of-work. Interactive and non-interactive variants of cost-functions can be constructed which can be used in situations where the server can issue a challenge (connection oriented interactive protocol), and where it can not (where the communication is store–and–forward, or packet oriented) respectively.

At the time of publication of [1] the author was not aware of the prior work by Dwork and Naor in [2] who proposed a CPU pricing function for the application of combatting junk email. Subsequently applications for cost-functions have been further discussed by Juels and Brainard in [3]. Jakobsson and Juels propose a dual purpose for the work spent in a cost-function: to in addition perform an otherwise useful computation in [4].

2 Cost-Functions

A *cost-function* should be efficiently verifiable, but parameterisably expensive to compute. We use the following notation to define a cost-function.

In the context of cost-functions we use *client* to refer to the user who must compute a *token* (denoted $\mathcal{T}$) using a cost-function MINT() which is used to create tokens to participate in a protocol with a *server*. We use the term *mint* for the cost-function because of the analogy between creating cost tokens and minting physical money.

The server will check the value of the token using an evaluation function VALUE(), and only proceed with the protocol if the token has the required value.

The functions are parameterised by the amount of work w that the user will have to expend on average to mint a token.

With *interactive cost-functions*, the server issues a challenge $\mathcal{C}$ to the client – the server uses the CHAL() function to compute the challenge. (The challenge function is also parameterised by the work factor.)

$$\begin{cases} \mathcal{C} \leftarrow \mathsf{CHAL}(s, w) & \text{server challenge function} \\ \mathcal{T} \leftarrow \mathsf{MINT}(\mathcal{C}) & \text{mint token based on challenge} \\ \mathcal{V} \leftarrow \mathsf{VALUE}(\mathcal{T}) & \text{token evaluation function} \end{cases}$$

With *non-interactive cost-functions* the client choses it's own challenge or random start value in the MINT() function, and there is no CHAL() function.

$$\begin{cases} \mathcal{T} \leftarrow \mathsf{MINT}(s, w) & \text{mint token} \\ \mathcal{V} \leftarrow \mathsf{VALUE}(\mathcal{T}) & \text{token evaluation function} \end{cases}$$

Clearly a *non-interactive cost-function* can be used in an interactive setting, whereas the converse is not possible.

2.1 Publicly Auditable, Probabilistic Cost

- A ***publicly auditable*** cost-function can be *efficiently* verified by any third party without access to any trapdoor or secret information. (When we say *publicly auditable* we mean implicitly that the cost-function is *efficiently* publicly auditable compared to the cost of minting the token, rather than auditable in the weaker sense that the auditor could repeat the work done by the client.)

- A ***fixed cost*** cost-function takes a fixed amount of resources to compute. The fastest algorithm to mint a fixed cost token is a deterministic algorithm.

- A ***probabilistic cost*** cost-function is one where the cost to the client of minting a token has a predictable expected time, but a random actual time as the client can most efficiently compute the cost-function by starting at a random start value. Sometimes the client will get lucky and start close to the solution.

  There are two types of probabilistic cost *bounded probabilistic cost* and *unbounded probabilistic cost*.

  - An ***unbounded probabilistic cost*** cost-function, can in theory take forever to compute, though the probablity of taking significantly longer than expected decreases rapidly towards zero. (An example would be the cost-function of being required to throw a head with a fair coin; in theory the user could be unlucky and end up throwing many tails, but in practice the probability of not throwing a head for k throws tends towards 0 rapidly as $\lim_{k\to\infty}(\frac{1}{2})^k = 0$.)
  - With a ***bounded probabilistic cost*** cost-function there is a limit to how unlucky the client can be in it's search for the solution; for example where the client is expected to search some key space for a known solution; the size of the key space imposes an upper bound on the cost of finding the solution.

2.2 Trapdoor-free

A disadvantage of known solution cost-functions is that the challenger can cheaply create tokens of arbitrary value. This precludes public audit where the server may have a conflict of interests, for example in web hit metering, where the server may have an interest to inflate the number of hits on it's page where it is being paid per hit by an advertiser.

- A ***trapdoor-free*** cost-function is one where the server has no advantage in minting tokens.

An example of a trapdoor-free cost-function is the Hashcash [1] cost-function. Juels and Brainard's client-puzzle cost-function is an example of a *known-solution* cost-function where the server has an advantage in minting tokens. Client-puzzles as specified in the paper are in addition not publicly auditable, though this is due to a storage optimization and not inherent to their design.

3 The Hashcash cost-function

Hashcash is a non-interactive, publicly auditable, trapdoor-free cost function with unbounded probabilistic cost.

First we introduce some notation: consider bitstring $s = \{0,1\}^\star$, we define $[s]_i$ to means the bit at offset i, where $[s]_1$ is the left-most bit, and $[s]_{|s|}$ is the right-most bit. $[s]_{i...j}$ means the bit-wise substring between and including bits i and j, $[s]_{i...j} = [s]_i \parallel ... \parallel [s]_j$. So $s = [s]_{1...|s|}$.

We define a binary infix comparison operator $\stackrel{\text{left}}{=}_b$ where b is the length of the common left-substring from the two bit-strings.

$$\begin{array}{ll} x \stackrel{\text{left}}{=}_0 y & [x]_1 \neq [y]_1 \\ x \stackrel{\text{left}}{=}_b y & \forall_{i=1...b}\ [x]_i = [y]_i \end{array}$$

Hashcash is computed relative to a service-name s, to prevent tokens minted for one server being used on another (servers only accept tokens minted using their own service-name). The service-name can be any bit-string which uniquely identifies the service (eg. host name, email address, etc).

The hashcash function is defined as (note this is an improved simplifed variant since initial publication see note in section 5:

$$\left\{ \begin{array}{ll} \mathsf{PUBLIC}: & \text{hash function } \mathcal{H}(\cdot) \text{ with output size } k \text{ bits} \\ & \\ \mathcal{T} \leftarrow \mathsf{MINT}(s,w) & \textbf{find } x \in_R \{0,1\}^\star \textbf{ st } \mathcal{H}(s\|x) \stackrel{\text{left}}{=}_w 0^k \\ & \textbf{return } (s,x) \\ \mathcal{V} \leftarrow \mathsf{VALUE}(\mathcal{T}) & \mathcal{H}(s\|x) \stackrel{\text{left}}{=}_v 0^k \\ & \textbf{return } v \end{array} \right.$$

The hashcash cost-function is based on finding *partial hash collisions* on the all 0 bits k-bit string 0^k. The fastest algorithm for computing partial collisions is brute force. There is no challenge as the client can safely choose his own random challenge, and so the hashcash cost-function is a *trapdoor-free* and *non-interactive* cost-function. In addition the Hashcash cost-function is *publicly auditable*, because anyone can efficiently verify any published tokens. (In practice $|x|$ should be chosen to be large enough to make the probability that clients reuse a previously used start value negligible; $|x| = 128$ bits should be enough even for a busy server.)

The server needs to keep a double spending database of spent tokens, to detect and reject attempts to spend the same token again. To prevent the database growing indefinitely, the service string can include the time at which it was minted. This allows the server to discard entries from the spent database after they have expired. Some reasonable expiry period should be chosen to take account of clock inaccuracy, computation time, and transmission delays.

Hashcash was originally proposed as a counter-measure against email spam, and against systematic abuse of anonymous remailers. It is necessary to use *non-interactive cost-functions* for these scenarios as there is no channel for the server to send a challenge over. However one advantage of *interactive cost-functions* is that it is possible to prevent pre-computation attacks. For example, if there is a cost associated with sending each email this may be sufficient to limit the scale of email abuse perpetrated by spammers; however for a pure DoS-motivated attack a determined adversary may spend a year pre-computing tokens to all be valid on the same day, and on that day be able to temporarily overload the system.

It would be possible to reduce the scope for such pre-computation attacks by using a slowly changing *beacon* (unpredictable broadcast authenticated value changing over time) such as say this weeks winning lottery numbers. In this event the current beacon value is included in the start string, limiting pre-computation attacks to being conducted within the time period between beacon value changes.

4 Interactive Hashcash

With the interactive form of hashcash, for use in interactive settings such as TCP, TLS, SSH, IPSEC etc connection establishment a challenge is chosen by the server. The aim of interactive hashcash is to defend server resources from premature depletion, and provide graceful degradation of service with fair allocation across users in the face of a DoS attack where one user attempts to deny service to the other users by consuming as many server resources as he can. In

the case of security protocols such as TLS, SSH and IPSEC with computationally expensive connection establishment phases involving public key crypto the server resource being defended is the servers available CPU time.

The interactive hashcash cost-function is defined as follows:

$$\begin{cases} \mathcal{C} \leftarrow \mathsf{CHAL}(s, w) & \textbf{choose } c \in_R \{0,1\}^k \\ & \textbf{return } (s, w, c) \\ \mathcal{T} \leftarrow \mathsf{MINT}(C) & \textbf{find } x \in_R \{0,1\}^\star \textbf{ st } \mathcal{H}(s\|c\|x) \stackrel{\text{left}}{=}_w 0^k \\ & \textbf{return } (s, x) \\ \mathcal{V} \leftarrow \mathsf{VALUE}(T) & \mathcal{H}(s\|c\|x) \stackrel{\text{left}}{=}_v 0^k \\ & \textbf{return } v \end{cases}$$

4.1 Dynamic throttling

With interactive hashcash it becomes possible to dynamically adjust the work factor required for the client based on server CPU load. The approach also admits the possibility that interactive hashcash challenge-response would only be used during periods of high load. This makes it possible to phase-in DoS resistent protocols without breaking backwards compatibility with old client software. Under periods of high load non-hashcash aware clients would be unable to connect, or would be placed in a limited connection pool subject to older less effective DoS counter-measures such as random connection dropping.

4.2 hashcash-cookies

With connection-slot depletion attacks such as the syn-flood attack, and straight-forward TCP connection-slot depletion the server resource that is being consumed is space available to the TCP stack to store per-connection state.

In this scenario it may be desirable to avoid keeping per connection state, until the client has computed a token with the interactive hashcash cost-function. This defense is similar to the syn-cookie defense to the syn-flood attack, but here we propose to additionally impose a CPU cost on the connecting machine to reserve a TCP connection-slot.

To avoid storing the challenge in the connection state (which itself consumes space) the server may choose to compute a keyed MAC of the information it would otherwise store and sent it to the client as part of the challenge so it can verify the authenticity of the challenge and token when the client returns them. (This general technique – of sending a record you would otherwise store together with a MAC to the entity the information is about – is referred to as a *symmetric key certificate*.) This approach is analogous to the technique used in syn-cookies, and Juels and Brainard proposed a related approach but at the application protocol level in their client-puzzles paper.

For example with MAC function $\mathcal{M}$ keyed by server key K the challenge MAC could be computed as:

$$\begin{cases} \mathsf{PUBLIC}: & \text{MAC function } \mathcal{M}(\cdot,\cdot) \\ \\ \mathcal{C} \leftarrow \mathsf{CHAL}(w) & \textbf{choose } c \in_R \{0,1\}^k \\ & \textbf{compute } m \leftarrow \mathcal{M}(K, t\|s\|p\|w\|c) \\ & \textbf{return } (t, s, p, w, c, m) \end{cases}$$

The client must send the MAC m, and the challenge c and challenge parameters p with the response token so that the server can verify the challenge and the response. The server should also include in the MAC the connection parameters, at minimum enough to identify the connection-slot and some time measurement or increasing counter t so that old challenge responses can not be collected and re-used after the connection-slots are free. The challenge and MAC would be sent in the TCP SYN-ACK response message, and the client would include the interactive hashcash token (challenge-response) in the TCP ACK message. As with syn-cookies, the server would not need to keep any state per connection prior to receiving the TCP ACK.

For backwards compatibility with syn-cookie aware TCP stacks, a hashcash-cookie aware TCP stack would only turn on hashcash-cookies when it detected that it was subject to a TCP connection-depletion attack. Similar arguments as given by Dan Bernstein in [5] can be used to show that backwards compatibility is retained, namely under syn-flood attacks Bernstein's arguments show how to provide backwards compatibility with non syn-cookie aware implementations; similarly under connection-depletion attack hashcash-cookies are only turned on at a point where service would anyway otherwise be unavailable to a non-hashcash-cookie aware TCP stack.

As the flood increases in severity the hashcash-cookie algorithm would increase the collision size required to be in the TCP ACK message. The hashcash-cookie aware client can still connect (albeit increasinly slowly) with a more fair chance against the DoS attacker presuming the DoSer has limited CPU resources. The DoS attacker will effectively be pitting his CPU against all the other (hashcash-cookie aware) clients also trying to connect. Without the hashcash-cookie defense the DoSer can flood the server with connection establishments and can more easily tie up all it's slots by completing n connections per idle connection time-out where n is the size of the connection table, or pinging the connections once per idle connection time-out to convince the server they are alive.

Connections will be handed out to users collectively in rough proportion to their CPU resources, and so fairness is CPU resource based (presuming each user is trying to open as many connections as he can) so the result will be biased in favor of clients with fast processors as they can compute more interactive-hashcash challenge-response tokens per second.

5 Hashcash improvements

In the initially published hashcash scheme, the target string to find a hash collision on was chosen *fairly* by using the hash of the service-name (and respectively the service-name and challenge in the interactive setting). A subsequent improvement suggested independently by Hal Finney [6] and Thomas Boschloo [7] for hashcash is to find a collision against a fixed output string. Their observation is that a fixed collision target is also fair, simpler and reduces verification cost by a factor of 2. A fixed target string which is convenient to compare trial collisions against is the k-bit string 0^k where k is the hash output size.

6 Low Variance

Ideally cost-function tokens should take a predictable amount of computing resources to compute. Juels and Brainard's client-puzzle construction provides a *probabilistic bounded-cost* by issuing challenges with *known-solutions*, however while this limits the theoretical worst case running time, it makes limited practical difference to the variance and typical experienced running time. The technique of using known solutions is also not applicable to the non-interactive setting. It is an open question as to whether there exist probabilistic bounded-cost, or fixed-cost non-interactive cost-functions with the same order of magnitude of verification cost as hashcash.

The other more significant incremental improvement due to Juels and Brainard is the suggestion to use multiple sub-puzzles with the same expected cost, but *lower variance* in cost. This technique should be applicable to both the non-interactive and interactive variants of hashcash.

6.1 Non-Parallelizability and Distributed DoS

Roger Dingledine, Michael Freedman and David Molnar put forward the argument that non-parallelizable cost-functions are less vulnerable to Distributed DoS (DDoS) in chapter 16 of [8]. Their argument is that non-parallelizable cost-functions frustrate DDoS because the attacker is then unable sub-divide and farm out the work of computing an individual token.

The author described a fixed-cost cost-function in [9] using Rivest, Shamir and Wagner's time-lock puzzle [10] which also happens to be non-parallelizable. The time-lock puzzle cost-function can be used in either an interactive or non-interactive setting as it is safe for the user to chose their own challenge. The applicability of Rivest et al's time-lock puzzle as a cost-function was also subsequently observed by Dingledine et al in [8].

For completeness we present the time-lock puzzle based fixed-cost and non-parallelizable cost-function from [9] here:

$$
\begin{cases}
\text{PUBLIC:} & n = pq \\
\text{PRIVATE:} & \text{primes } p \text{ and } q,\ \phi(n) = (p-1)(q-1) \\
 & \\
\mathcal{C} \leftarrow \mathsf{CHAL}(s, w) & \textbf{choose } c \in_R [0, n) \\
 & \textbf{return } (s, c, w) \\
\mathcal{T} \leftarrow \mathsf{MINT}(\mathcal{C}) & \textbf{compute } x \leftarrow \mathcal{H}(s \| c) \\
 & \textbf{compute } y \leftarrow x^{x^w} \pmod{n} \\
 & \textbf{return } (s, c, w, y) \\
\mathcal{V} \leftarrow \mathsf{VALUE}(\mathcal{T}) & \textbf{compute } x \leftarrow \mathcal{H}(s \| c) \\
 & \textbf{compute } z \leftarrow x^w \bmod \phi(n) \\
 & \textbf{if } x^z = y \pmod{n} \textbf{ return } w \\
 & \textbf{else return } 0
\end{cases}
$$

The client does not know $\phi(n)$, and so the most efficient method for the client to calculate MINT() is repeated exponentiation, which requires w exponentiations. The challenger knows $\phi(n)$ which allows a more efficient computation by reducing the exponent $\bmod \phi(n)$, so the challenger can execute VALUE() with 2 modular exponentiations. The challenger as a side-effect has a trapdoor in computing the cost-function as he can compute MINT() efficiently using the same algorithm.

We argue however that the added DDoS protection provided by non-parallelizable cost-functions is marginal: unless the server restricts the number of challenges it hands out to a recognizably unique client the DDoS attacker can farm out multiple challenges as easily as farm out a sub-divided single challenge, and consume resources on the server at the same rate as before. Further it is not that hard for a single client to masquerade as multiple clients to a server.

Consider also: the DDoS attacker has generally due to the nature of his method of commandeering nodes an equal number of network connected nodes at his disposal as processors. He can therefore in any case have each attack node directly participate in the normal protocol indistinguisably from any legitimate user. This attack strategy is also otherwise optimal anyway as the attack nodes will present a varied set of source addresses which will foil attempts at per-connection fairness throttling strategies and router based DDoS counter-measures based on volume of traffic across IP address ranges. Therefore for the natural attack node marshalling patterns non-parallelizable cost-functions offer limited added resistance.

As well as the arguments against the practical efficacy and value of non-parallelizable cost-functions, to date non-parallelizable cost functions have had orders of magnitude slower verification functions than non-parallelizable cost-functions. This is because the non-parallelizable cost-functions so far discussed in the literature are related to trap-door public key cryptography constructs which are inherently less efficient. It is an open question as to whether there exist non-parallelizable cost-functions based on symmetric-key (or public-key) constructs with verification functions of the same order of magnitude as those of symmetric-crypto based cost-functions.

While for the application of time-lock puzzles to cost-functions, a reduced public key size could be used to speed up the verification function, this approach introduces risk that the modulus will be factored with the result that the attacker gains a big advantage in minting tokens. (Note: factoring is itself a largely parallelizable computation.)

To combat this the server should change the public parameters periodically. However in the particular case of the public parameters used by time-lock puzzles (which are the same as the RSA modulus used in RSA encryption), this operation is itself moderately expensive, so this operation would not be performed too frequently. It would probably not be wise to deploy software based on key sizes below 768 bits for this aplication, in addition it would help to change keys periodically, say every hour or so. (RSA modulii of 512 bits have recently been factored by a closed group as discussed in [11] and more recently have been demonstrated by Nicko van Someren et al to be factorizable using standard equipment in an office as reported in [12]; DDoS attackers are known be able to muster significant resources, probably easily exceeding those used in this demonstration.)

The time-lock puzzle cost-function also is necessarily trap-door as the server needs a private verification-key to allow it to efficiently verify tokens. The existance of a verification-key presents the added risk of key compromise allowing the attacker to by-pass the cost-function protection. (The interactive hashcash cost-function by comparison is trap-door-free, so there is no key which would allow an attacker a short-cut in computing tokens). In fact if the verification-key were compromised, it could be replaced, but this need adds complexity and administrative overhead as this event needs to be detected and manual intervention or some automated detection triggering key-replacement implemented.

The time-lock puzzle cost-function also will tend to have larger messages as there is a need to communicate planned and emergency re-keyed public parameters. For some applications, for example the syn-cookie and hashcash-cookie protocols, space is at a premium due to backwards compatibility and packet size constraints imposed by the network infrastructure.

So in summary we argue that non-parallelizable cost-functions are of questionable practical value in protecting against DDoS attacks, have more expensive verification functions, incur the risk of verification key compromise and attendant key management complexities, have larger messages, and are significantly more complex to implement. We therefore recommend instead the simpler hashcash protocol (or if the public-auditability and non-interactive options are not required Juels and Brainard's client-puzzles are roughly equivalent).

7 Applications

Apart from the initially proposed applications for hashcash of throttling DoS against remailer networks and detering email spam, since publication the following applications have been discussed, explored and in some cases implemented and deployed:

- hashcash-cookies, a potential extension of the syn-cookie as discussed in section 4.2 for allowing more graceful service degradation in the face of connection-depletion attacks.
- interactive-hashcash as discussed in section 4 for DoS throttling and graceful service degradation under CPU overload attacks on security protocols with computationally expensive connection establishment phases. No deployment but the analogous client-puzzle system was implemented with TLS in [13]
- hashcash throttling of DoS publication floods in anonymous publication systems such as Freenet [14], Publius [15], Tangler [16],
- hashcash throttling of service requests in the cryptographic Self-certifying File System [17]
- hashcash throttling of USENET flooding via mail2news networks [18]
- hashcash as a minting mechanism for Wei Dai's b-money electronic cash proposal, an electronic cash scheme without a banking interface [19]

8 Cost-function classification scheme

We list here a classification of characteristics of cost-functions. We use the following notation to denote the properties of a cost-function:

$$([e = \{1, \tfrac{1}{2}, 0\}], [\sigma = \{0, \tfrac{1}{2}, 1\}], [\{i, \bar{i}\}], [\{a, \bar{a}\}], [\{t, \bar{t}\}], [\{p, \bar{p}\}])$$

Where e is the efficiency: value $e = 1$ means *efficiently-verifiable* – verifiable with cost comparable to or lower than the cost of verifying symmetric key constructs such as hashcash which consume just a single compression round of an iterative compression function based hash function such as SHA1 or MD5. Value $e = \frac{1}{2}$ means *practically-verifiable* we mean less efficiently than *efficienty-verifiable*, but still efficient enough to be practical for some applications, for example the author considers the time-lock puzzle based cost-function with it's two modular exponentiations to fall into this category. Value $e = 0$ means *verifiable but impractical*, that the cost-function is verifiable but the verification function is impractically slow such that the existance of the cost-function serves only as a proof of concept to be improved upon for practical use.

And σ is a characterization of the standard-deviation, value $\sigma = 0$ means *fixed-cost*, $\sigma = \frac{1}{2}$ means *bounded probabilistic cost* and $\sigma = 1$ means *unbounded probabilistic cost*. Note by *bounded probabilistic-cost* we mean usefully bounded – a bound in the work factor in excess of a work-factor that an otherwise functionally similar unbounded cost-function would only reach with negligible probability would not be useful.

And i denotes that the cost-function is *interactive*, and $\bar{i}$ that the cost-function is *non-interactive*.

And a denotes that the cost-function is *publicly auditable*, $\bar{a}$ denotes that the cost-function is not *publicly auditable*, which means in practice that it is only verifiable by the service using a private key material. Note by *public-auditability*

we mean *efficiently* publicly-auditable, and would not consider repeating the work of the token minter as adequate efficiency to classify.

And t denotes that the server has a *trapdoor* in computing the cost-function, conversely $\bar{t}$ denotes that server has no trapdoor in computing the cost-function.

And p denotes that the cost-function is parallelizable, $\bar{p}$ deontes that the cost-function is *non-parallelizable*.

	trapdoor-free	trapdoor
interactive	hashcash $(e = 1, \sigma = 1, i, a, \bar{t}, p)$	client-puzzles $(e = 1, \sigma = \frac{1}{2}, i, a, t, p)$ time-lock $(e = \frac{1}{2}, \sigma = 0, i, \bar{a}, t, \bar{p})$
non-interactive	hashcash $(e = 1, \sigma = 1, \bar{i}, a, \bar{t}, p)$	time-lock $(e = \frac{1}{2}, \sigma = 0, \bar{i}, \bar{a}, t, \bar{p})$

8.1 Open Problems

- existance of *efficiently-verifiable non-interactive fixed-cost* cost-functions $(e = 1, \sigma = 0, \bar{i})$ (and the related weaker problem: existance of same with *probabilistic bounded-cost* $(e = 1, \sigma = \frac{1}{2}, \bar{i})$)
- existance of *efficiently-verifiable non-interactive non-parallelizable* cost-functions $(e = 1, \bar{i}, \bar{p})$ (and the related weaker problem: existance of same in interactive setting $(e = 1, i, \bar{p})$)
- existance of *publicly-auditable non-interactive fixed-cost* cost-functions $(\sigma = 0, \bar{i}, a)$ (and the related weaker problem: existance of same with *bounded probabilistic-cost* $(\sigma = \frac{1}{2}, \bar{i}, a)$)

References

[1] Adam Back. Hashcash, May 1997. Published at http://www.cypherspace.org/hashcash/.

[2] Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In *Proceedings of Crypto*, 1992. Also available as http://www.wisdom.weizmann.ac.il:81/Dienst/UI/2.0/Describe/ncstrl.weizmann_il/CS95-20.

[3] Ari Juels and John Brainard. Client puzzles: A cryptographic countermeasure against connection depletion attacks. In *Network and Distributed System Security Symposium*, 1999. Also available as http://www.rsasecurity.com/rsalabs/staff/bios/ajuels/publications/client-puzzles/.

[4] Markus Jakobsson and Ari Juels. Proofs of work and bread pudding protocols. In *Proceedings of the IFIP TC6 and TC11 Joint Working Conference on Communications and Multimedia Security (CMS '99), Leuven, Belgium*, September 1999. Also available as http://citeseer.nj.nec.com/238810.html.

[5] Dan Bernstein. Syn cookies. Published at http://cr.yp.to/syncookies.html.

[6] Hal Finney. Personal communication, Mar 2002.

[7] Thomas Boschloo. Personal communication, Mar 2002.

[8] Andy Oram, editor. *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*. O'Reilly and Associates, 2001. Chapter 16 also available as http://freehaven.net/doc/oreilly/accountability-ch16.html.

[9] Adam Back. Hashcash - amortizable publicly auditable cost functions. Early draft of paper, 2000.

[10] Ronald L Rivest, Adi Shamir, and David A Wagner. Time-lock puzzles and timed-release crypto. Technical Report MIT/LCS/TR-684, 1996. Also available as http://theory.lcs.mit.edu/~rivest/publications.html.

[11] Herman te Riele. Security of e-commerce threatened by 512-bit number factorization. Published at http://www.cwi.nl/~kik/persb-UK.html, Aug 1999.

[12] Dennis Fisher. Experts debate risks to crypto, Mar 2002. Also available as http://www.eweek.com/article/0,3658,s=720&a=24663,00.asp.

[13] Drew Dean and Adam Stubblefield. Using cleint puzzles to protect tls. In *Proceedings of the 10th USENIX Security Symposium*, Aug 2001. Also available as http://www.cs.rice.edu/~astubble/papers.html.

[14] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore Hong. Freenet: A distributed anonymous information storage and retrieval system. In Hannes Federrath, editor, *Proceedings of the International Workshop on Design Issues in Anonymity and Unobservability*. Springer, 2001. Also available as http://freenetproject.org/cgi-bin/twiki/view/Main/Papers.

[15] Marc Waldman, Aviel D Rubin, and Lorrie Faith Cranor. Publius: A robust, tamper-evident, censorship-resistant web publishing system. In *Proceedings of the 9th USENIX Security Symposium*, Aug 2000. Also available as http://www.usenix.org/publications/library/proceedings/sec2000/waldman/waldman_html/v2.html.

[16] Marc Waldman and David Mazieres. Tangler: A censorship resistant publishing system based on document entanglement. In *Proceedings of the 8th ACM Conference on Computer and Communication Security*, Nov 2001. Also available as http://www.cs.nyu.edu/~waldman/.

[17] David Mazieres. *Self-certifying File System*. PhD thesis, Massachusetts Institute of Technology, May 2000. Also available as http://scs.cs.nyu.edu/~dm/.

[18] Alex de Joode. Hashcash support at dizum mail2news gateway. Published at https://ssl.dizum.com/hashcash/, 2002.

[19] Wei Dai. b-money. Published at http://www.eskimo.com/~weidai/bmoney.txt, Nov 1998.

Secure Names for Bit-Strings

Stuart Haber*
stuart@surety.com

W. Scott Stornetta*
scotts@surety.com

Abstract

The increasing use of digital documents, and the need to refer to them conveniently and unambiguously, raise an important question: can one "name" a digital document in a way that conveniently enables users to find it, and at the same time enables a user in possession of a document to be sure that it is indeed the one that is referred to by the name? One crucial piece of a complete solution to this problem would be a method that provides a cryptographically verifiable label for any bit-string (for example, the content, in a particular format, of the document). This problem has become even more acute with the emergence of the World-Wide Web, where a document (whose only existence may be on-line) is now typically named by giving its URL, which is merely a pointer to its virtual location at a particular moment in time.

Using a one-way hash function to call files by their hash values is cryptographically verifiable, but the resulting names are unwieldy, because of their length and randomness, and are not permanent, since as time goes on the hash function may become vulnerable to attack. We introduce procedures to create names that are short and meaningful, while at the same time they can persist indefinitely, independent of the longevity of any given hash function. This is done by naming a bit-string according to its position in a growing, directed acyclic graph of one-way hash values. We prove the security of our naming procedures under a reasonable complexity-theoretic cryptographic assumption, and then describe practical uses for these names. An implementation of our naming scheme has been in use since January 1995.

1 Introduction

Users of documents need to refer to those documents in order to keep records and in order to communicate with other users of the documents. In practice, users name their documents in various ways. A name must be unambiguous, at least in the context of its use; this requires some connection between the name and the integrity of the document it names.

In the traditional world of paper documents, there are usually reasonable guarantees of this connection. In the case of printed books and magazines, large print runs that are the result of single typesetting efforts make it easier to be confident that all copies of a printed document are the same, with a definite name printed in a conventional place in the document. Making a change to a paper document of any sort, even a small change, typically leaves forensic evidence.

A characteristic feature of digital documents, by contrast, is that they are easy to copy and to alter. The naming problem is especially troubling if the document exists only on-line and never in conventional paper-based form. For on-line documents, a useful naming scheme would allow users to employ the name to *find* documents, as well as to check the *integrity* of the documents that they find. A number of proposals have been made for such naming systems (see e.g. [SM 94, KW 95, BD+ 95]). These proposals address in different ways the problem of how to "resolve" the name into a location where the document might be found.

It is the integrity-checking problem that we address in this work: how to make sure that the bit-string content of a given digital document is indeed the same as the bit-string that was intended. Heretofore, two different sorts of mechanisms have been proposed, digital signatures and one-way hash values.

Having the author or publisher of a document compute a digital signature for its bit-string content is a reasonable use of cryptographic tools for this purpose. (See, for example, [R 95, M 94].) However, the ability to validate many digital signatures requires the presence of a public-key infrastructure, and the trustworthiness of the validation procedure relies on the assurance that the signer's private signing key is indeed secure. For some on-line documents, the infrastructure and these assurances may not be available. For long-lived documents, the security of the binding between a public key and the person or role of the putative signer becomes even more problematic. (A general solution to the latter problem is briefly described in §5.)

Thus it would be useful to have an integrity mechanism, depending on the exact contents of the bit-string in question, that does not depend on the secrecy of a cryptographic key. A natural choice for such a mechanism is the use of a one-way hash function, naming any bit-string by its hash value. (See, for example, [BD+ 95].) However, while this method is intrinsically verifiable, there are several inconvenient features:

- A desirable feature for the names given to a collection

*Surety Technologies, 1 Main Street, Chatham, N.J. 07928, U.S.A.

CCS 97, Zurich, Switzerland

of objects is that they be long-lasting, if not permanent. (This is one of the functional requirements for URNs [SM 94].) But as technology advances, any particular choice of a presumably one-way function for a naming scheme becomes less secure, so that it must be replaced (see [Dob 96a, Dob 96b]).[1] The unpleasant result is that the name of a long-lived document will need to change over time.

- Hash values are too long for a human user to remember or even to communicate easily to another human being. (For example, it is currently recommended that one-way hash functions compute outputs that are at the very least 128 bits long; this is the output length of MD5 [Riv 92]. In a 6 bit/character encoding, this is 22 alphanumeric characters long.)
- The author of a bit-string document has no control over the form of its name. A one-way hash function produces a random-appearing bit-string of the appropriate length as the hash value of a document. Thus, inconvenient as it may be for the author, there will be no connection between the names of documents that are related to each other, either in form or in substance.

This paper presents a method for naming bit-strings that retains the verifiable security of hash-based names, while avoiding the constraints listed above, as well as avoiding the use of secret cryptographic keys. The method is a variation on the digital time-stamping schemes of [HS 91, BHS 93]. In summary, the essence of the new scheme is to keep a repository of hash values that depend on many bit-string inputs, and to name each bit-string by a concise description of a location in the repository to which it can be securely "linked" by a one-way hashing computation.

An implementation of our naming scheme has been running continuously since January 1995 [Sur 95].

The rest of this paper is organized as follows. After technical preliminaries in §2, including both a brief discussion of the wider problem of naming digital documents as well as a formal description of our sub-problem, we present our scheme and prove its security in §3. Motivated by the explosive growth of the Internet, we mention a number of possible applications of our scheme in §4. In §5, we describe a method for extending the lifetime of our digital names beyond the cryptographically secure lifetime of the hash functions used to compute them. Finally, we discuss several different sorts of practical implementation in §6.

2 Preliminaries

2.1 Naming digital documents

A naming system for digital documents should perform (at least) two functions. It should help the user (1) to find the document named; and (2) to reassure himself or herself that a given document is indeed the correct one, *i.e.* that it is indeed a perfect copy of the document that was intended.

To enable both these functions, the "name" could include both identification information as well as location information. System design may include procedures for registration of new documents, for finding a document given its name, for updating a document's location information, and for validating the integrity of a document. Typically, there is a server that "resolves" or translates a name into location information, for example into a URL or a list of URLs. The name may include other information about the document, including such data as title, author, format, price, and access privileges.

A large body of work has been devoted to the difficult problem of designing and building a naming system of this sort so that it is usable, useful, and reliable. In [SM 94] a set of functional requirements is described for Uniform Resource Names (URNs), the names to be assigned by a naming system for resources on the Internet. A number of researchers have built naming systems, including, among others, [KW 95, BD+ 95]. (This is by no means an exhaustive list.)

In this work we propose a new method for the integrity-checking piece of naming systems for digital documents. All previously proposed systems that included mechanisms for checking the integrity of the bit-string or bit-strings that make up a digital document have used either digital signatures or one-way hash functions for this purpose. For certain applications, these methods have the problems described in §1 above.

2.2 Hash functions

The principal technical tool we use in this paper is that of a one-way hash function. This is a function compressing digital documents of arbitrary length to bit-strings of a fixed length, for which it is computationally infeasible to find two different documents that are mapped by the function to the same *hash value*. (Such a pair is called a *collision* for the hash function.)

Practical proposals for one-way hash functions include those of MD5 [Riv 92], SHA-1 [NIST 94], and RIPEMD-160 [DBP 96]. Though the actual security of these functions (*i.e.*, the precise difficulty of computing collisions for them) is not known, they are now in more or less widespread use.

Definition In a more theoretical vein, Damgård defined a family of *collision-free hash functions* to be a family $\{H_k\}_k$ of sets of functions (indexed by a security parameter k) with the following properties:

1. Each H_k is a set of functions $h : \{0,1\}^* \to \{0,1\}^k$ that are computable in polynomial time.
2. Given k, it is easy to choose $h \in H_k$ at random.
3. It is computationally infeasible, given a random choice of one of these functions, to find a collision for the function. More precisely, for any polynomial algorithm A, for any positive constant c,

   $$\Pr[h \leftarrow H_k; (x, x') \leftarrow A(h) : x \neq x', h(x) = h(x')] < k^{-c}$$

   for sufficiently large k.

Damgård gave a constructive proof of their existence, on the assumption that there exist families of one-way "claw-free" permutations [Dam 87]. More generally, any "one-way group action" is sufficient [BY 90]. Concretely, the construction can be based on the difficulty either of factoring or of the discrete logarithm function. (As usual, the collision adversary A in condition (3) above can be uniform or non-uniform, depending on the precise form of the hypothesis made on the computational complexity of the underlying problem.) For

[1] For example, because of recent attacks on MD5, RSA Laboratories recommends that "in the future MD5 should no longer be implemented in signature schemes, where a collision-resistant hash function is required" [Dob 96c].

a variety of reasons, none of the known theoretical constructions of collision-free hash functions are practical.

In practice, the infeasibility of computing collisions for a particular hash function depends on the current state of the art, both the current state of algorithmic knowledge about attacking the function in question, as well as the computational speed and memory available in the best current computers. As the state of the art advances, it is likely that a function that was once securely one-way will eventually cease to be so. For example, Dobbertin's recently announced attacks on MD4 and MD5 have considerably reduced the community's confidence in the strength of these two functions [Dob 96a, Dob 96b, Dob 96c]. In §5 below we offer a solution to the problem this poses for certain practical systems whose real-world security depends on the actual infeasibility of specific computational tasks.

We refer the reader to [Pre 93] for a thorough discussion of one-way hash functions.

2.3 Theoretical model

We emphasize that this is a theoretical description of the problem of verifiably "naming" bit-strings, which is only a piece of the larger problem of naming digital documents.

The setting for our problem is a distributed network of parties. The network may include a *server* S as well as a *repository* R; parties may query the repository, asking for a copy of a particular item it contains.

Definition A *naming scheme* for this setting consists of:

- a *security parameter* k;
- a polynomial-time *naming protocol* N, possibly requiring interaction with the server S, taking as input a bit-string x, and producing as output a *name* n for x, a *certificate* c, and the addition of items to the repository R; and
- a polynomial-time *validation protocol* V, that takes as input a triple (x, n, c) and the result of a query to R, and either accepts or rejects its inputs.

If (n, c) is the output of an invocation of N on input x, then V accepts the input (x, n, c) when it is accompanied by a correct response to a query to R.

It is possible, of course, to specify a naming scheme that does not require a server or a repository. In that case, the naming protocol and the validation protocol may simply be algorithms that any party in the network may invoke without interacting with outside parties.

Definition A *counterfeiting adversary* to a naming scheme $[N, V, S]$ is a (possibly probabilistic) algorithm A that performs as follows. Given k as input, A produces (polynomially many) naming requests $x_1, x_2, \ldots$; for each x_i A is given the output of $N(x_i)$. The request x_{i+1} may be computed after A has received the response to its ith request. In addition, A may make (polynomially many) queries to R. Finally (after q naming requests, say), A's output is of the form (x, n, c). This output is a *successful counterfeit* if $x \neq x_i$ (for $i = 1 \ldots q$) and V accepts (x, n, c) (after a correct response to any queries to R).

Definition A naming scheme is *secure* if for any polynomially bounded counterfeiting adversary A and for any positive constant c, A's success probability on input k is less than k^{-c} for sufficiently large k.

To illustrate our definitions, here is a simple example of a naming scheme, where the only role of the server is to announce its random choice of a hash function $h \in H_k$. The naming procedure is just $N(x) = h(x)$ with no certificates, and V accepts (x, n) if $n = h(x)$. It is clear that this defines a secure naming scheme as long as H_k is the kth set in a family of collision-free hash functions.

We remark that the roles of S as trusted server and R as trustworthy repository in these definitions are just an artifact of how we have chosen to present and to analyze our naming schemes, allowing a clean separation between issues of the security of the scheme itself and issues of how it might be implemented in practice.

2.4 Digital time-stamping

Our solution to the naming problem builds on the work of [HS 91] and [BHS 93], whose authors describe several procedures with which users can *certify* (the bit-string contents of) their digital documents, computing for any particular document a time-stamp *certificate*. Later, any user of the system can *validate* a document-certificate pair; that is, he or she can use the certificate to verify that the document existed, in exactly its current form, at the time asserted in the certificate. It is infeasible to compute an illegitimate document-certificate pair that will pass the validation procedure.

Because we use it directly in our naming scheme, we summarize here one digital time-stamping scheme. A central "coordinating server" receives *certification* requests—essentially, hash values of files—from users. At regular intervals, the server builds a binary tree out of all the requests received during the interval, following Merkle's tree authentication technique; the leaves are the requests, and each internal node is the hash of the concatenation of its two children [Merk 80]. The root of this tree is hashed together with the previous "interval hash" to produce the current interval hash, which is placed in a widely available *repository*. The server then returns to each requester a time-stamp *certificate* consisting of the time at which the interval ended, along with the list of sibling hash values along the path leading from the requester's leaf up to the interval hash, each one accompanied by a bit indicating whether it is the right or the left sibling. The scheme also includes a *validation* procedure, allowing a user to test whether a document has been certified in exactly its current form, by querying the repository for the appropriate interval hash, and comparing it against a hash value appropriately recomputed from the document and its certificate.

It is noteworthy that the trustworthiness of the certificates computed in this scheme depends only on the integrity of the repository, and not (for example) on trusting that a particular private key has not been compromised or that a particular party's computation has been performed correctly.

3 A naming scheme for bit-strings

Next we describe a naming scheme for a network that includes a server S and a repository R. Many executions of N and of V may be performed concurrently in the network. We assume that there exists a family $\{H_k\}_k$ of collision-free hash functions. Given an initial choice of security parameter k, S announces to all parties its random choice of a one-way hash function $h \in H_k$. Our scheme is a variation on the time-stamping scheme described in §2.4 above, with

S playing the role of the coordinating server that computes certificates in response to requests and makes additions to the repository R.

We abbreviate a bit-string's certificate by omitting the list of hash values, leaving only a pointer to the relevant interval hash (for example, the time at which it was computed), and an encoding of the position of the request in the tree for that interval (for example, the sequence of left or right bits). It is this abbreviation that we propose to use as the name of the bit-string.

More explicitly, an invocation of N on input x begins with the computation of $y = h(x)$, and the submission of y to S, which includes y as one of the leaves of the tree being built in the current time interval. At the end of the interval, having built a tree of height l (that includes the previous interval hash), S places the root of the tree in R as the current interval hash with label t, say. S responds to the request by returning the certificate $c = [t; (z_1, b_1), \ldots, (z_l, b_l)]$, where each $b_i = \mathsf{L}$ or R. Finally, the name returned by N for argument x is $n = [t; b_1, \ldots, b_l]$.

One uses the entire certificate in order to validate that a particular string correctly names a particular bit-string document, first by checking that the putative name was correctly extracted from the certificate, and then by following the usual validation procedure for the document-certificate pair (recomputing the path from the leaf to the root of the tree).

To be precise, V operates as follows, given as inputs a document x, a name $n = [t; b_1, \ldots, b_l]$, and a certificate $c = [t'; (z_1, b'_1), \ldots, (z_l, b'_l)]$: First, V checks that $t = t'$ and that each $b_i = b'_i$. Next, V computes $y_1 \leftarrow h(x)$ and then (for $i \leftarrow 1 \ldots l$) if $b_i = \mathsf{L}$ then $y_{i+1} \leftarrow h(z_i \cdot y_i)$ else if $b_i = \mathsf{R}$ then $y_{i+1} \leftarrow h(y_i \cdot z_i)$. Finally, V queries R for the hash value stored at location t, and checks that it is identical to y_{l+1}. V accepts if all these checks are satisfied and rejects otherwise.

Figure 1 below illustrates the tree built by S for a time interval during which it received eight requests, containing the eight hash values a, b, c, d, e, f, g, and h. In this diagram, ab is the hash of the concatenation of a and b, etc., and IH_t and IH_{t-1} are the respective interval hashes for the current and the previous intervals. The certificate computed by S for the third request (the one containing hash value c), for example, is the following:

$$[t;\ (d, \mathsf{R}), (ab, \mathsf{L}), (eh, \mathsf{R}), (IH_{t-1}, \mathsf{L})].$$

3.1 Security

The security of this naming scheme follows directly from the infeasibility of computing hash collisions for functions from $\{H_k\}_k$, since the only possible counterfeit names include hash collisions. In essence, if x is a bit-string on which N was never invoked during a run, any triple (x, n, c) that V will accept (after the correct response to a query to R) will include a hash collision for the function h announced by S at the beginning of the run: either x itself or one of the hash values z_i in c (when combined on the left or the right with y_i) collides with another argument to h whose hash value was computed during the run. Therefore we have the following theorem.

Theorem 1 *If $\{H_k\}_k$ is a family of collision-free hash functions, then the naming scheme $[N, V, S]$ described above is secure.*

Because the reduction in the proof is so direct, it is easy to give an "exact security" analysis (cf. [Lev 85, BKR 94]) of the strength of this scheme, whether the hash functions used are from the collision-free family provided by a theoretical cryptographic assumption or rather practical hash functions, as in the implementations described in §6 below.

3.2 Variations on the scheme

Of course, the secure verifiability of the names assigned by the scheme described above does not depend on the particular combination of binary trees and linked lists used. By systematically invoking the hash function on pairs or ordered lists of hash values, new hash values can be computed from old ones so as to form a directed acyclic graph (by directing an edge from each of the inputs to the hash value output). Design considerations (including those discussed in §6.1 below) may dictate several different combinatorial structures for this directed graph.

Whatever the structure of the growing graph of hash values, it is secured by making portions of the graph widely witnessed and widely available. To insure the verifiability of the names, it suffices that every document in the naming structure be linked by a directed path to a widely witnessed hash value; a standard ordering of the incoming edges at each node can be used to encode the path. Then the name of a document is given by this encoding of its location in the graph, together with a pointer to the hash value at the end of the path, and the argument of Theorem 1 applies.

For example, in one variation of the scheme described above, a list of documents may be used to build a local tree (following Merkle, again), whose root is sent off in turn as a request to the coordinating server. The location information for a document in this "tree-of-trees" scheme can be written as a position in the server's tree followed by a position in the local tree.

In another variation, the widely witnessed hash values in the repository could consist simply of a linked list (as in the simple linking scheme of [HS 91]). In this case the location information for a document is a simple pointer into the repository.

4 Applications

The problem of naming digital documents might have seemed like a curiosity only a few years ago. However, with the growth in use of the Internet, more and more people need to be able to refer confidently to meaningful bit-sequences. The problem is now a matter of immediate practical concern.

The problem has become especially acute with the emergence of the World-Wide Web. Jumping from one URL (Uniform Resource Locator) to the next in a sequence of WWW documents may seem at first to be exactly analogous to following a bibliographic reference in a traditional scholarly paper. In fact it is something quite different: a URL is only a pointer to a location, with no guarantee that what a user finds there today is the same reference that the author originally intended. If on-line citations include secure names for the bit-string contents of the documents cited, then it is possible to traverse a path of citations with confidence that one is indeed following the authors' intentions. This ability would be especially useful for the many documents on the World-Wide Web that exist only on-line.

In most electronic commerce systems, transaction records of all sorts are kept on-line, and it would be useful to have a cryptographically secure means of assigning serial numbers or tracking numbers to these records.

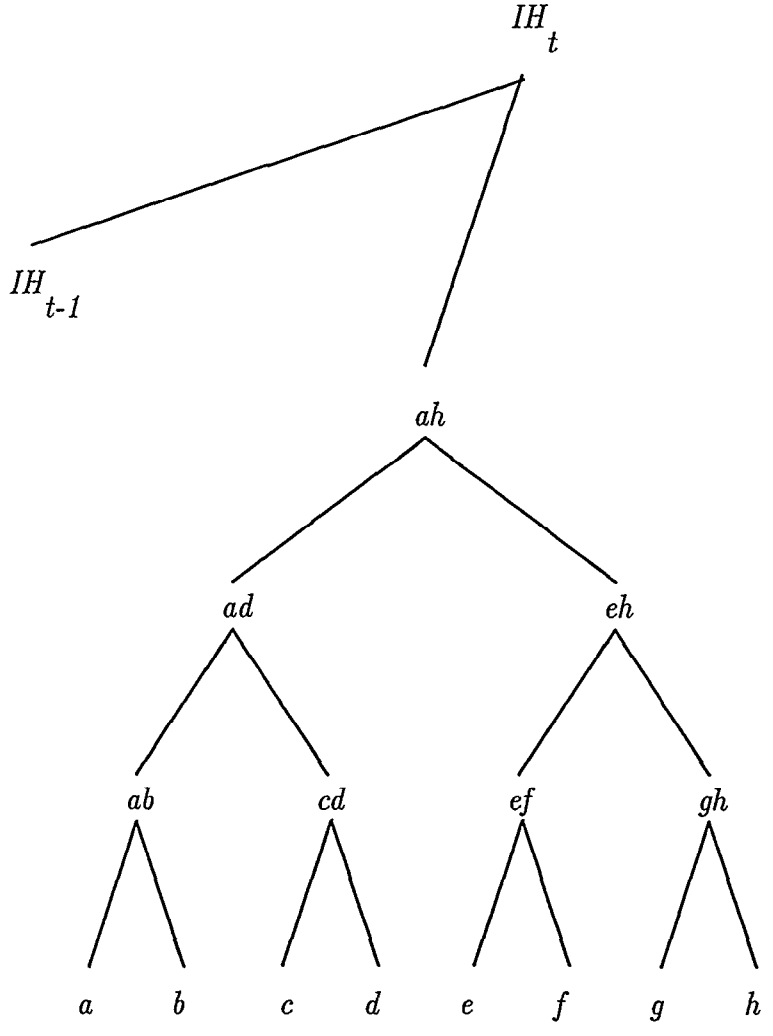


Figure 1: 8-leaf tree for the example of §3.

Software code is another class of digital document for which it would be useful to have an easy way for a short name to carry a guarantee of integrity. A user who downloads software (along with its naming certificate) from a site on the Net can be sure of its integrity if he or she is able to check that the code is correctly named by a short string of letters and numbers. Here, of course, bit-string equality is *exactly* the point. The great strength of using secure names in this application is that the short name of a program is considerably easier to distribute widely and robustly than the program itself. (It is also easier to distribute reliably than the sort of public-key infrastructure information that is required in order to use digital signatures in order to validate the integrity of code.)

For another example of a type of large digital document whose integrity matters a great deal, consider the case of genetic data. Scientists now routinely download others' data sets for use in their own research. The use of our naming scheme would allow the user to be sure of the data's integrity, as well as providing a convenient and verifiable way to cite the data in published descriptions of the work that was done with it.

5 Long-lived names

The technique described in [BHS 93] for renewing cryptographic certifications of authenticity applies directly to the certificates of the present naming scheme.

The renewing process works as follows. Let us suppose that an implementation of a particular time-stamping system is in place, and consider the pair (x, C), where C is a valid time-stamp certificate (in this implementation) for the bit-string x. Now suppose that an improved time-stamping system is implemented and put into practice—by replacing the hash function used in the original system with a new hash function, or even perhaps after the invention of a completely new algorithm. Further suppose that the pair (x, C) is time-stamped by the new system, resulting in a new certificate C', and that some time later, *i.e.* at a definite later date, the original method is compromised. C' provides evidence not only that the document contents x existed prior to the time of the new time-stamp, but that it existed at the time stated in the original certificate, C; prior to the compromise of the old implementation, the only way to create a certificate was by legitimate means. (It is similarly recommended that if a digitally signed document is likely to be important for a long time—perhaps longer than the signer's key will be valid—then the document-signature pair should be time-stamped [BHS 93, Odl 95, HKS 95].)

In our naming schemes, the verifiable name for the bit-string x is a standard abbreviation a for its original certificate C. In order that a continue to be verifiable as a name for x, the certificate C should be renewed (as above) from time to time as new time-stamping systems are put in place. As long as this is done, a is still a verifiable name for x. There is now an additional step to the procedure for validating the name: after checking that a is correctly extracted from C, one must follow the usual time-stamp validation procedure for the certificate, which now includes both the original-system validation of (x, C) and the new-system validation of $[(x, C), C']$. We note that in practice this additional validation step would be automated, and would not at all affect the convenient use of a to name x.

6 Practical implementations

A practical implementation of a naming scheme cannot use the known theoretical constructions of collision-free hash functions. If the decision is made to use practical one-way hash functions such as MD5, then users of the system do not need to trust the server's random choice of a function $h \in H_k$. (However, they do have to hope that the hash function chosen is one-way in practice; see section §5 for one way to allay users' concerns on this score.)

The naming scheme described in §3 above, based on the digital time-stamping scheme described in §2.4, was implemented by Surety Technologies, and has been in continuous commercial use since January 1995. The implementation uses practical hash functions; specifically, the current implementation uses $h(x) = (\mathrm{MD5}(x), \mathrm{SHA}(x))$ as the hash value for any argument x. A number of supplemental mechanisms are employed in order to maintain the integrity and wide distribution of the repository [Sur 95].

The names assigned by our scheme are indeed concise, growing essentially as slowly as possible while still providing unique names. If the repository contains n interval hashes, and no more than m naming requests are received during each interval, the names can be written with at most $\lg_2 nm$ bits. Just to give a numerical example, a repository representing a thousand requests per minute for the length of a century requires 36-bit names; in the MIME encoding (six bits per alphanumeric character) such a name can be jotted down with six characters, while hash-value names of this length are completely insecure.

6.1 Meaningful names

There are several variations of our naming scheme that allow an author a fair measure of control over the names of his or her documents, so that the author can choose a verifiable name that is meaningful in one or another useful way.

First, and most obviously, observe that in the scheme described in detail in §3 a convenient way to encode the location in the repository to which a document's contents are linked is by the date and time at which the interval hash at that location was computed. Instead of (*e.g.*) a MIME encoding of the number of seconds since a moment in early 1970 (Unix standard time), it would often be useful to express at least a part of this date and time in human-readable form.

In a slight variation, we can allow "personalized" naming requests, as follows. Suppose that the repository items are formatted in a standard way every day, and let $F(\cdot)$ denote any standard mapping from ASCII-encoded strings to the list of daily repository locations. When the server receives a personalized naming request that includes the ASCII string s, the request is held until the appropriate moment in the day and then linked to the widely witnessed hash value stored at location $F(s)$; in this way, s is made to be part of the name of the documents included in those special naming requests. Thus, for example, the author of *The History of Computers in Zurich* can arrange for the verifiable name of its bit-string contents to have the form ["The History of Computers in Zurich" `date suffix`], where `suffix` includes a few bits of disambiguating information that distinguishes this request from all others that were linked to the same repository location.

In another example, consider the tree-of-trees variation briefly mentioned in §3.2. An author can name a multi-part document by placing the contents of each successive part at

consecutive leaf nodes of a local tree. The resulting request to the server gives the consecutive parts of the document consecutive local positions and therefore consecutive names. Furthermore, the other portions of these consecutive names are identical, explicitly encoding the fact that they are parts of the same document. And local trees can have sub-trees, so that our historian can arrange to name the ith section of the jth chapter of his masterpiece ["The History of Computers in Zurich" `infix` $i.j$], for all appropriate pairs (i, j).

More complicated ways of structuring the parts of a document can similarly be encoded in the verifiable names assigned by our naming scheme. Note that conventional naming schemes do allow for encoding document structure into names, but not in a verifiable manner.

In another variation, a table of contents for a long or complicated multi-part document can be included in a standard place in the request—for example, as its last piece. The table of contents may contain more or less detailed descriptions of the parts of the document. At a later time, together with a list of documents to be authenticated and their certificates, such an authenticated table of contents can be used to verify (1) that each document in the list is an exact copy of one that was registered with the table of contents, and (2) that none of the documents in the list are missing.

Acknowledgements

We would like to thank Ralph Merkle, R. Venkatesan, Matt Franklin, Avi Rubin, Bill Arms, and Dave Richards for helpful discussions about this work. We would also like to thank the anonymous referees for their very useful suggestions.

References

[BHS 93] D. Bayer, S. Haber, and W.S. Stornetta. Improving the efficiency and reliability of digital time-stamping. In *Sequences II: Methods in Communication, Security, and Computer Science*, ed. R.M. Capocelli, A. De Santis, U. Vaccaro, pp. 329–334, Springer-Verlag, New York (1993).

[BKR 94] M. Bellare, J. Kilian, and P. Rogaway. The security of cipher block chaining. In *Advances in Cryptology—Crypto '94*, Lecture Notes in Computer Science, Vol. 839, ed. Y. Desmedt, pp. 94-107, Springer-Verlag (1994).

[BY 90] G. Brassard and M. Yung. One-way group actions. In *Advances in Cryptology—Crypto '90*, Lecture Notes in Computer Science, Vol. 537, pp. 94-107, Springer-Verlag (1991).

[BD+ 95] S. Browne, J. Dongarra, S. Green, K. Moore, T. Pepin, T. Rowan, and R. Wade. Location-independent naming for virtual distributed software repositories. Univeristy of Tennessee Computer Science TR 95-278 (1995). (Available at `http://www.cs.utk.edu/~library/TechReports/1995/`).

[Dam 87] I. Damgård. Collision-free hash functions and public-key signature schemes. In *Advances in Cryptology—Eurocrypt '87*, Lecture Notes in Computer Science, Vol. 304, pp. 203–217, Springer-Verlag (Berlin, 1988).

[Dob 96a] H. Dobbertin. Cryptanalysis of MD4. In *Fast Software Encryption*, Lecture Notes in Computer Science, Vol. 1039, ed. D. Gollman, pp. 53–69, Springer-Verlag (Berlin, 1996).

[Dob 96b] H. Dobbertin. Cryptanalysis of MD5 compress. Private communication (May 1996). Described by B. Preneel, Rump Session, Eurocrypt '96 (May 1996).

[Dob 96c] H. Dobbertin. The status of MD5 after a recent attack. *CrytoBytes*, Vol. 2, No. 2 (Summer 1996).

[DBP 96] H. Dobbertin, A. Bosselaers, and B. Preneel. RIPEMD-160: A strengthened version of RIPEMD. In *Fast Software Encryption*, Lecture Notes in Computer Science, Vol. 1039, ed. D. Gollman, pp. 71–82, Springer-Verlag (Berlin, 1996).

[HKS 95] S. Haber, B. Kaliski, and W.S. Stornetta. How do digital time-stamps support digital signatures? *CryptoBytes*, Vol. 1, No. 3 (Autumn 1995). (Available at `http://www.rsa.com/rsalabs/pubs/cryptobytes.html`.)

[HS 91] S. Haber and W.S. Stornetta. How to time-stamp a digital document. *Journal of Cryptology*, Vol. 3, No. 2, pp. 99–111 (1991).

[KW 95] R. Kahn and R. Wilensky. A framework for distributed digital object services. Corporation for National Research Initiatives technical report cnri.dlib/tn95-01 (May 1995). (Available at `http://www.cnri.reston.va.us/`.)

[Lev 85] L.A. Levin. One-way functions and pseudorandom generators. In *Proceedings of the 17th Annual Symposium on Theory of Computing*, pp. 363-365, ACM (1987).

[Merk 80] R.C. Merkle. Protocols for public key cryptosystems. In *Proc. 1980 Symposium on Security and Privacy*, IEEE Computer Society, pp. 122–133 (April 1980).

[M 94] J.W. Moore. The use of encryption to ensure the integrity of reusable software components. In *Proc. 3rd International Conf. on Software Reusability*, IEEE Computer Society Press (November 1994).

[NIST 94] National Institute of Standards and Technology. Secure Hash Standard. NIST Federal Information Processing Standard Publication 180-1 (May 1994).

[Odl 95] A. Odlyzko. The future of integer factorization. *CrytoBytes*, Vol. 1, No. 2 (1995).

[Pre 93] B. Preneel. *Analysis and Design of Cryptographic Hash Functions.* Ph.D. dissertation, Katholieke Universiteit Leuven (January 1993).

[Riv 92] R. Rivest. The MD5 Message-Digest Algorithm. Internet Network Working Group Request for Comments 1321 (April 1992).

[R 95] A. Rubin. Trusted distribution of software over the Internet. In *Internet Society 1995 Symposium on Network and Distributed System Security* (1995).

[SM 94] K. Sollins and L. Masinter. Functional requirements for Uniform Resource Names. Internet Network Working Group Request for Comments 1737 (December 1994).

[Sur 95] Surety Technologies, Inc. Answers to Frequently Asked Questions about the Digital Notary™ System. `http://www.surety.com` (since January 1995).

PROTOCOLS FOR PUBLIC KEY CRYPTOSYSTEMS

Ralph C. Merkle

ELXSi International

Sunnyvale, Ca.

Abstract

New cryptographic protocols which take full advantage of the unique properties of public key cryptosystems are now evolving. Several protocols for public key distribution and for digital signatures are briefly compared with each other and with the conventional alternative.

1. Introduction

The special strengths of public key systems are briefly considered by examining cryptographic protocols for key distribution and digital signatures using both public key and conventional systems.

The reader is assumed to be familiar with the general ideas behind public key cryptosystems, as described in [1,10].

For many of the following examples we assume there are two communicants, called A and B, and an opponent E. A and B will attempt to send secret messages and sign contracts, while E will attempt to discover the keys, learn the secrets, and forge contracts. Sometimes, A will attempt to evade a contract he signed with B, or B will attempt to forge A's signature to a new contract.

A and B will need to apply one way functions to various arguments of various sizes, so we assume we have a one way function F which can be applied to arguments of any size and produce a fixed size output. For a more complete discussion of one way functions, see [2,9,13,19].

This work was partially supported under NSF Grant ENG 10173, and much of the work was done at Stanford University ISL. The author would also like to acknowledge the support of BNR Inc, where much of the work reported here was done. An extended version has been submitted to CACM.

CH1522-2C/80/0000-0122$00.75

Symposium on Security and Privacy.

2. Centralized Key Distribution

Centralized key distribution using conventional encryption functions was the only reasonable method of handling key distribution in a multi-user network environment before the discovery of public key distribution methods. Only conventional encryption functions need be used, which presently offers a performance advantage. (Presently known public key systems are less efficient than conventional cryptographic systems. Whether or not this will continue is not now known. Discovery of new public key systems seems almost inevitable, and discovery of more efficient ones probable.)

In centralized key distribution, A, B, and all other system users somehow deposit a conventional cryptographic key with a central key distribution center. If A wishes to communicate with B, the key distribution center will send a common (session) key to A and B using the previously agreed on central keys. A and B can then communicate with no further assistance from the key distribution center.

This protocol is simple and requires only conventional encryption functions. Its use has been defended in the literature [17,18,20].

The major drawback of this protocol is its vulnerability to both centralized loss of security and centralized loss of function. Theft of the central keys, or bribery of personnel at the central site will compromise all users of the system. Similarly, destruction of the central keys destroys the key distribution mechanism for all users.

The security and reliability of centralized key distribution can be increased by using two or more centers, each with its own keys [1]. Destruction or compromise of a single center will not affect the other centers.

Security can also be improved if all the user keys are encrypted with a master key by the center. The master key must still be stored securely (and suitable provision made for its backup), but the (encrypted) user keys can be stored anywhere. This approach is used by IBM [23].

This protocol does not fully solve the key distribution problem: some sort of key distribution method must be used between each user and the center to establish the original keys. This problem is nontrivial because no electronic communications can be used and inexpensive physical methods, e.g., registered mail, offer only moderate security. The use of couriers is reasonably secure, although more expensive.

3. Simple Public Key Distribution

This is the most basic application of public key systems [1,5,6,7,8]. Its purpose is to allow A and B to agree on a common key k without any prior secret arrangements, even though E overhears all messages. A randomly computes enciphering and deciphering keys E_A and D_A, and sends E_A to B (and E). B picks the random key, k, and transmits $E_A(k)$ to A (and E). A computes $D_A(E_A(k)) = k$. A then discards both E_A and D_A, and B discards E_A. The key in future communications is k. It is used to encrypt all further messages using a conventional encryption function. Once A and B have finished talking, they both discard k. If they later resume the conversation the process is repeated to agree on a new key k'.

This protocol is very simple, and has a great deal to recommend it. First, no keys and no secret materials exist before A and B start communicating, and nothing is retained after they have finished. It is impossible for E to compromise any keys either before the conversation takes place, or after it is over, for the keys exist only during the conversation.

The disadvantage of this protocol is that E might actively interfere with the exchange of keys. Worse yet, E can force a known k on both A and B.

4. Authenticated Public Key Distribution

The now classic protocol [1] for secure and authenticated communications between A and B is: A and B generate E_A and E_B and make them public, while keeping D_A and D_B secret. The public enciphering keys of all users are entered in a public file, allowing easy and authenticated access to E_X for any user, X.

If A and B wish to agree on a common key k, then each sends a (session) key to the other by encrypting it with the others public key. The two keys thus agreed on are combined and used to encrypt further messages.

At the end of this protocol, A and B have agreed on a common key, k, which is both secret and authenticated.

This protocol suffers from two weaknesses. First, entries in the public file might be altered. This can be dealt with both by good physical security, or by using new protocols (see sections 5 and 6) for authenticating the entries in the public file.

Second, secret deciphering keys can be lost. This problem must ultimately be solved by good physical security.

5. Public Key Distribution with Certificates

Kohnfelder [3] first suggested that entries in the public file be authenticated by having a Central Authority (CA) sign them with D_{CA}. He called such signed entries certificates.

The protocol with certificates is the same as the authenticated protocol, except that A and B can now check the entries in the public file by checking each other's certificates. This protocol assures A and B that each has the other's public enciphering key, and not the public enciphering key of some imposter.

The security of this protocol rests on the assumptions that the secret deciphering keys of A, B, and CA have not been compromised; that A and B have correct copies of E_{CA} (to check the signed certificates); and that CA has not issued a bad certificate, either deliberately because it was untrustworthy, or accidentally because it was tricked.

E_{CA} can be published in newspapers and magazines, and sent over all available communication channels: blocking its correct reception would be very difficult.

If D_{CA} is compromised, then it is no longer possible to authenticate the users of the system and their public enciphering keys. The certificates are now worthless because the (unauthorized) person who has learned D_{CA} can produce false certificates at will.

6. Public Key Distribution with Tree Authentication

Key distribution with certificates was vulnerable to the criticism that D_{CA} can be compromised, resulting in system wide loss of authentication. This problem can be solved by using tree authentication [13].

Again, this protocol attempts to authenticate entries in the public file. However, instead of signing each entry in the public file, this protocol applies a one way hash function, H, to the entire public file. Even though H is applied to the entire public file, the output of H is only 100 or 200 bits long. The (small) output of H will be called the root, R, of the public file.

If all users of the system know R, then all users can authenticate the correctness of the (whole) public file by computing R = H(public file). Any attempt to introduce changes into the public file will imply R ≠ H(altered public file), an easily detected fact.

This method effectively eliminates the possibility of compromising D_{CA} because no secret deciphering key exists.

Because the public file will be subjected to the harsh glare of public scrutiny, and because making alterations in the public file is effectively impossible after it has been published, a high degree of assurance that it is correct can be attained.

This method is impractical as stated. Fortunately, it is possible to selectively authenticate individual entries in the public file without having to know the whole public file by using Merkle's "tree authentication," [13].

The essence of tree authentication is to authenticate the entire public file by "divide and conquer." If we define $\underline{Y}$ = public file = $Y_1, Y_2, \ldots Y_n$, (so the $\underline{i}$th entry in the public file is denoted Y_i, and B's entry is Y_B); we can define H(public file) = H($\underline{Y}$) as:

H($\underline{Y}$) = F(H(first half of $\underline{Y}$),
H(second half of $\underline{Y}$))

Where F is a one way function.

If A wishes to confirm B's public enciphering key, then A need only know the first half of the public file, (which is where Y_B appears) and H(second half of public file) which is only 100 bits long. A can compute H(public file) knowing only this information, and yet A only knew half the entries in the public file.

In a similar fashion, A does not really need to know all of the first half of the public file, for

H(first half of public file) =
F(H(first quarter of public file),
H(second quarter of public file))

All A needs to know is the first quarter of the public file (which has Y_B), and H(second quarter of public file).

By applying this concept recursively, A can confirm Y_B in the public file knowing only R, $\log_2$ n intermediate values, and Y_B itself. The information needed to authenticate Y_B, given that R has already been authenticated, lies along the path from R to Y_B and will be called the authentication path.

These definitions are illustrated in figure 1, which shows the authentication path for Y_5.

For a more detailed discussion the reader is referred to [13].

Using tree authentication, user A has an authentication path which can be used to authenticate user A's public enciphering key, provided only that R has already been authenticated. An "authentication path" is a new form of certificate, with E_{CA} replaced by R.

This protocol can only be compromised if: D_A or D_B is compromised, or if R is not correctly known by A or B, or if there is a false and misleading entry in the public file.

The latter two are easily detectable. If either A or B has the wrong R, they will be unable to complete the protocol with any other legitimate user who has the correct R, a fact that will be quickly detected.

Because the public file is both open to public scrutiny and unalterable, false or misleading entries can be rapidly detected. In practice, a few users concerned with correctness can verify that the public file satisfies some simple global properties, i.e., each user name appears once and once only in the entire public file; individual users can then verify that their own entry is correct, and need not bother examining the rest of the public file.

The only practical method of compromising this protocol is to compromise D_A or D_B. A user's security is thus dependent on himself and no one else.

7. Digital Signatures

The use of public key cryptosystems to provide digital signatures was suggested by Diffie and Hellman [1]. Rivest, Shamir and Adleman [8] have suggested an attractive implementation. Signature techniques based on methods other than public key cryptosystems have been suggested by Lamport and Diffie [1,24], Rabin [15], and Merkle [13].

Digital signatures, whether based on conventional encryption functions, on public key cryptosystems, on probabilistic computations, or on other techniques share several important properties in common. These common properties are best illustrated by the following now classic example.

A wishes to place a purchase order with his stock broker B. A, on the Riviera, cannot send a written order to B in New York in time. All that A can quickly send to B is information, i.e., a sequence of bits, but B is concerned that A may later disclaim the order. A

must somehow generate a sequence of bits (a digital signature) which will convince B (and if need be a judge) that A authorized the order. It must be easy for B to validate the digital signature, but impossible for him (or anyone other than A) to generate it (to prevent charges that B was dabbling in the market illegally with A's money).

There are digital signature schemes which do not involve public key cryptosystems but it will be convenient notationally to let A sign message m by computing the signature, $D_A(m)$. Checking a signature will then be done by computing $m = E_A(D_A(m))$. If $E_A(D_A(m))$ produces an illegible message (random bits) then the signature is rejected as invalid. This notation is somewhat misleading because the actual method of generating and validating signatures can be very different from this model; it is retained because it is widely known and because we will not discuss the differences among different digital signature methods, only their common properties.

Digital signature protocols are naturally divided into three parts: a method of signing messages used by A, a method for authenticating a signature used by B, and a method for resolving disputes, used by the judge. It is important to note that two protocols that differ only in the method of resolving disputes are different. Failure to understand this point has led to confusion in the literature [17,20].

We now turn to specific digital signature protocols.

8. A Conventional Signature Protocol

A conventional "signature" protocol relies on the observation that if A and B trust some central authority CA, and if A and B have a secure method of communicating with CA, then A can "sign" a message simply by sending it to CA and relying on CA to adjudicate disputes. This approach is defended by some [17].

This protocol is subject to the weaknesses of centralized key distribution (described earlier).

9. The Basic Digital Signature Protocol

The first public key based digital signature protocol [1], proceeded by having A sign message m by computing $D_A(m)$ and giving it to B as the signed

message. B (or a judge) can compute $E_A(D_A(m)) = m$, thus confirming the correctness of the signed message. A is held responsible for a signed message if and only if it can be verified by applying A's public enciphering key to it.

This protocol can be criticized [16,17,20] on two grounds: First, the public file might have been tampered with. Methods of authenticating the public file, discussed previously under key distribution protocols, solve this problem.

A second criticism is that A has no recourse should his secret deciphering key be compromised and made public. Anyone can sign any message they desire with A's compromised D_A, and A will be held responsible.

It seems clear that A will only agree to this digital signature protocol if he can provide very good physical security for D_A. The loss to A if D_A is compromised can be substantial.

A different method of solving this problem is to alter the dispute resolution protocol so that A is not held responsible for his signature if his secret deciphering key is compromised and made public.

The fact that altering the dispute resolution procedure creates a different protocol has not been fully appreciated, and the preceding two protocols have been confused with each other for this reason. Some criticism of "the" public key digital signature protocol has actually been of this second protocol, and failed to consider the first protocol at all.

If we assume that A knows D_A, then under the second protocol A can make D_A public and effectively disavow the signed message. For this reason, some critics have argued that this protocol is inadequate.

If we assume that A does not know D_A, then he is unable to disavow his signature under this protocol. It is easy to design a system in which this is the case.

The major difference between the second protocol and the first is in the division of risk: in the second protocol B will be left holding the bag if A's signing key is compromised. Clearly, B must be given assurances that this condition is unlikely before he will be willing to use this protocol.

10. The Time-Stamp Protocol

A protocol that would allow A to report loss or theft of D_A and disclaim messages signed after the reported loss yet force A to acknowledge the validity of signatures made before the reported loss must involve the concept of time.

We introduce time into the following protocol by using time-keepers who can digitally time-stamp information given to them. We assume that both A and B have agreed on a set of acceptable time-keepers whose time-stamps will be accepted in dispute resolution.

If A can report that D_A has been lost, then he must report this fact to some agent who will be responsible for answering queries about the current status of D_A, i.e., has it been lost or not. For simplicity, we shall assume this role is played by the central authority, CA. CA will sign messages stating that A's secret deciphering key has not been compromised as of the current time. These signed messages will be called "validity-checks."

In the time stamp protocol, user A signs message m by computing $D_A(m)$ and sending it to B. B then has a time-keeper time stamp the message and obtains a validity-check from CA. If D_A has already been reported lost B rejects the signature, otherwise he accepts.

In dispute resolution, the judge holds that a message has been validly signed if and only if it can be checked by applying A's public enciphering key AND it has been time-stamped prior to any reported loss of D_A.

This protocol provides very good assurance to all parties that they have been dealt with fairly.

The major disadvantage of this protocol, as compared with the basic digital signature protocol, is the requirement that B obtain both a time-stamp and a validity-check, presumably in real time. These requirements force the use of a communications network, which both increases expense and decreases reliability.

If B is willing to obtain the time-stamp and the validity-check after the transaction has been completed, i.e., within a few days, an off-line system can be used. This modified protocol could be used by B either as a fail-soft protocol during communications outages, or as the standard protocol if communication costs are too high.

Off-line operation is cheaper and more reliable, but it exposes B to some risk: A might have recently reported the loss of D_A and B would not know about it. If physical security for secret deciphering keys is good, this risk should be minimal.

11. Witnessed Digital Signatures

If the value of a transaction is high enough, it might be desirable to have a witness physically confirm that A

signed message m. The witness, W, would compute D_W("I, W, physically saw A agree to and sign message m."). It would be necessary for A and B to agree in advance on acceptable witnesses.

The primary advantage of this protocol is that it reduces B's risk. The primary disadvantage is that it forces A to find a (physically present) witness to confirm the transaction.

12. Digital Signature Applications Not Involving Dispute

Not all applications of digital signatures involve contracts between two potentially disputing parties. Digital signatures are also an ideal method of broadcasting authenticated messages from a central source which must be confirmed by many separate recipients, or repeatedly confirmed by the same recipient at different times to insure that the message has not been modified.

One example of such an application is the distribution of network software to individual nodes of a communications network. It would be clearly undesirable for any node to start executing the wrong software. On the other hand, it is very desirable to send updates to the nodes over the network itself. The obvious solution is for updates to be digitally signed by an appropriate network administrator, and for the nodes to check the digital signature prior to executing them.

This example leads naturally to another application of digital signatures in operating system security. A major risk to the security of an operating system is the possibility that the system code that it is executing today is not the same that it was executing yesterday: someone might have put a trap door into the operating system that lets them do anything they please. To guard against this possibility, the operating system could refuse to execute any code in privileged mode unless that code had been properly signed. Carried to its logical conclusion, the operating system would check the digital signature of privileged programs each time they were loaded into central memory If this check were implemented in hardware, it would be impossible for any software changes to subvert it. The machine would be physically incapable of executing code in privileged mode unless that code was signed.

If privileged programs are digitally signed by the programmer who originally wrote them, as well as by various supervisory levels, and if the computer is physically unable to execute unsigned

code in privileged mode, then it is possible to have complete assurance that the privileged programs running on the computer right now have not been modified since they were given there final checkout and signed by the programmer. Of course, this does not necessarily mean that the operating system is secure, but it does eliminate a major class of worries.

13. Conclusions

This paper has briefly described a number of cryptographic protocols. Certainly, these are not the only ones possible; however, they are valuable tools to the system designer: they illustrate what can be achieved and provide feasible solutions to problems of recurring interest.

Further constructive work in this area is very much needed.

14. ACKNOWLEDGEMENTS

It is a great pleasure for the author to acknowledge the pleasant and informative conversations he had with Dov Andelman, Whitfield Diffie, Martin Hellman, Raynold Kahn Loren Kohnfelder, Frank Olken, and Justin Reyneri.

15. BIBLIOGRAPHY

1. Diffie, W., and Hellman, M. New directions in cryptography. IEEE Trans. on Inform. IT-22, 6(Nov. 1976), 644-654.

2. Evans A., Kantrowitz, W., and Weiss, E. A user authentication system not requiring secrecy in the computer. Comm. ACM 17, 8(Aug. 1974), 437-442.

3. Kohnfelder, L.M. Towards a practical public-key cryptosystem. MIT EE Bachelor's thesis.

4. Lipton, S.M., and Matyas, S.M. Making the digital signature legal--and safeguarded. Data Communications (Feb. 1978), 41-52.

5. McEliece, R.J. A public-key cryptosystem based on algebraic coding theory. DSN Progress Report, JPL, (Jan. and Feb. 1978), 42-44.

6. Merkle, R. Secure Communications over Insecure Channels. Comm. ACM 21, 4(Apr. 1978), 294-299.

7. Merkle, R., and Hellman, M. Hiding information and signatures in trapdoor knapsacks. IEEE Trans. on Inform. IT-24, 5(Sept. 1978), 525-530.

8. Rivest, R.L., Shamir, A., and Adleman, L. A method for obtaining digital signatures and public-key cryptosystems. Comm. ACM 21, 2(Feb. 1978), 120-126.

9. Wilkes, M.V., Time-Sharing Computer Systems. Elsevier, New York, 1972.

10. Diffie, W., and Hellman, M.E., Privacy and authentication: an introduction to cryptography, Proceedings of the IEEE Vol. 67, No. 3, Mar. 1979 pp. 397-427.

11. Squires, J. Russ monitor of U.S. phones, Chicago Tribune pp. 123, June 25, 1975.

12. Davis, R. Remedies sought to defeat Soviet eavesdropping on microwave links, Microwave Syst., vol. 8, no. 6, pp. 17-20, June 1978.

13. Merkle, R.C. A certified digital signature, to appear, CACM.

14. Kahn, D. The Codebreakers, New York: Macmillan. 1967.

15. Rabin, M.O., Digitalized signatures, in Foundations of Secure Computation, ed. Demillo, R.A., et. al. pp. 155-166.

16. Saltzer, J. On Digital Signatures, private communication.

17. Popek G.J. and Kline, C.S. Encryption Protocols, Public Key Algorithms, and Digital Signatures in Computer Networks; in Foundations of Secure Computation pp. 133-153.

18. Needham R.M. and Schroeder, M.D. Using Encryption for Authentication in Large Networks of Computers. CACM 21,12 Dec. 1978 pp. 993-999.

19. Merkle, R. Secrecy, authentication, and public key systems. Stanford Elec. Eng. Ph.D. Thesis, ISL SEL 79-017, 1979.

20. Popek, G.J., and Kline, C.S. Encryption and Secure Computer networks. Computing Surveys 11,4 Dec. 1979 pp. 331-356.

21. Simmons, G.J. Symmetric and Asymmetric Encryption. Computing Surveys 11,4 Dec. 1979 pp. 305-330.

22. Lamport, L. Time, clocks, and the ordering of events in a distributed system. CACM 21,7 Jul 1978 pp. 558-565.

23. Ehrsam, W.F., Matyas, S.M., Meyer, C.H., and Tuchman, W.L. A cryptographic key management scheme for implementing the data encryption standard. IBM Sys. Jour. 17,2 1978 pp. 106-125.

24. Lamport, L., Constructing digital signatures from a one way function. SRI Intl. CSL - 98

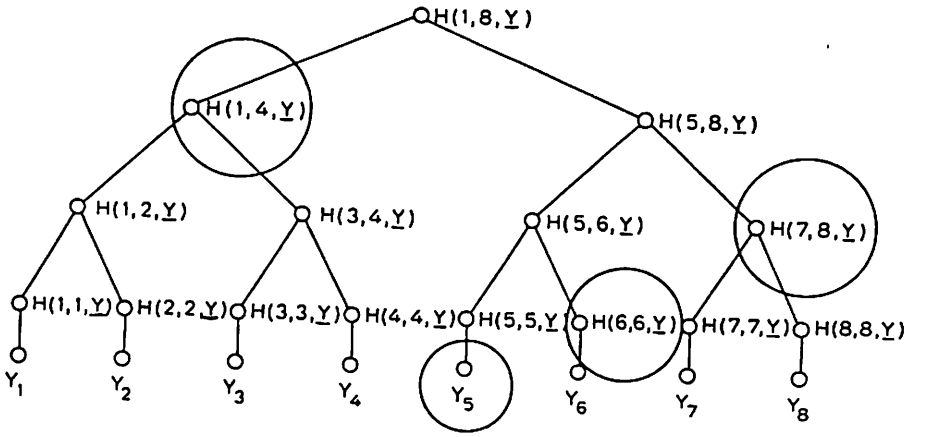


FIG. 1

An Introduction to Probability Theory and Its Applications

WILLIAM FELLER
Eugene Higgins Professor of Mathematics
Princeton University

VOLUME I

SECOND EDITION

John Wiley & Sons, Inc.
New York · London

tical lines $x = \frac{3}{2}$ and $x = n - \frac{1}{2}$. Now $\frac{1}{2}\log n$ quite obviously exceeds the area of the strip $n - \frac{1}{2} < x < n$ under the curve, and hence a_n *exceeds* the area under the curve and between $x = \frac{3}{2}$ and $x = n$. In other words, we have shown that

$$\int_{\frac{3}{2}}^{n} \log x \cdot dx < a_n < \int_{1}^{n} \log x \cdot dx. \tag{9.4}$$

The indefinite integral of $\log x$ is given by $x \log x - x$, and equation (9.4) reduces to the double inequality

$$(n + \tfrac{1}{2}) \log n - n + \tfrac{3}{2}(1 - \log \tfrac{3}{2}) < \log n! < (n + \tfrac{1}{2}) \log n - n + 1. \tag{9.5}$$

Put for abbreviation

$$\delta_n = \log n! - (n + \tfrac{1}{2}) \log n + n. \tag{9.6}$$

Then $1 - \delta_n$ is the difference between the extreme right member of (9.5) and $\log n!$, that is, $1 - \delta_n$ equals the area of the domain between the curve $y = \log x$ and the polygon $A_1A_2 \ldots A_n$. It follows that δ_n *decreases monotonically*. But by (9.5) we have $\frac{3}{2}(1 - \log \frac{3}{2}) < \delta_n < 1$. We conclude that δ_n tends to a limit comprised between 1 and $\frac{3}{2}(1 - \log \frac{3}{2})$. Denoting this limit by $\log c$ we have

$$\delta_n \to \log c \qquad \text{where} \quad 2.45 < c < 2.72. \tag{9.7}$$

In logarithmic notation Stirling's formula reduces to (9.7) with $c = (2\pi)^{\frac{1}{2}}$ (or 2.507, approximately). Now π can be defined in many ways, and for our purposes it is simplest and most natural to *define* $\pi = c^2/2$. With this definition we have Stirling's formula, but it remains to show that the constant so defined agrees with the more familiar π of other formulas. This fact will develop as a by-product of other calculations in chapter VII, and so the proof of Stirling's formula will be completed there.

Refinements. Stirling's formula can be improved by the addition of further terms. Although we shall never make use of such refinements, we shall here indicate the proof of the following *double inequality* [15]

$$(2\pi)^{\frac{1}{2}} n^{n+\frac{1}{2}} e^{-n+1/(12n+1)} < n! < (2\pi)^{\frac{1}{2}} n^{n+\frac{1}{2}} e^{-n+1/(12n)}. \tag{9.8}$$

To prove (9.8) note that

$$\delta_n - \delta_{n+1} = \left(n + \frac{1}{2}\right) \log \frac{n+1}{n} - 1 = \frac{1}{3(2n+1)^2} + \frac{1}{5(2n+1)^5} + \cdots \tag{9.9}$$

[15] H. Robbins, A remark on Stirling's formula, *American Mathematical Monthly*, vol. 62 (1955), pp. 26–29.

[the last expansion follows from (8.11) on setting $t = 1/(2n+1)$]. We *increase* the extreme right member in (9.9) by replacing the coefficients $\frac{1}{5}, \frac{1}{7}, \frac{1}{9}, \ldots$ by $\frac{1}{3}$; this leads to a geometric series with ratio $(2n+1)^{-2}$, and thus

$$\delta_n - \delta_{n+1} < \frac{1}{3[(2n+1)^2 - 1]} = \frac{1}{12n} - \frac{1}{12(n+1)}. \tag{9.10}$$

Accordingly, $\delta_n - 1/12n$ *increases* monotonically. Now the limit of this sequence is given by Stirling's formula, and passing to antilogarithms we have the second inequality in (9.8). The first inequality follows similarly from (9.9) on noticing that

$$\delta_n - \delta_{n+1} > \frac{1}{3(2n+1)^2} > \frac{1}{12n+1} - \frac{1}{12(n+1)+1}. \tag{9.11}$$

The accuracy of the approximations (9.8) is remarkable; even for $n = 1$ the formula leads to the two bounds 0.9958... and 1.0023.... The upper bound provided in (9.8) is slightly better [cf. (12.28)]. For $n = 2$ it yields 2.0007, for $n = 5$ we get 120.01..., and for $n = 10$ the first five significant figures are correct.

PROBLEMS FOR SOLUTION

Note: Sections 11 and 12 contain problems of a different character and diverse complements to the text.

10. EXERCISES AND EXAMPLES

Note: *Assume in each case that all arrangements have the same probability.*

1. How many different sets of initials can be formed if every person has one surname and (*a*) exactly two given names, (*b*) at most two given names, (*c*) at most three given names?

2. In how many ways can two rooks of different colors be put on a chessboard so that they can take each other?

3. Letters in the Morse code are formed by a succession of dashes and dots with repetitions permitted. How many letters is it possible to form with ten symbols or less?

4. Each domino piece is marked by two numbers. The pieces are symmetrical so that the number-pair is not ordered. How many different pieces can be made using the numbers 1, 2, ..., n?

5. The numbers 1, 2, ..., n are arranged in random order. Find the probability that the digits (*a*) 1 and 2, (*b*) 1, 2, and 3, appear as neighbors in the order named.

6. (*a*) Find the probability that among three random digits there occur 2, 1, or 0 repetitions. (*b*) Do the same for four random digits.

7. Find the probabilities p_r that in a sample of r random digits no two are equal. Estimate the numerical value of p_{10}, using Stirling's formula.

8. What is the probability that among k random digits (*a*) 0 does not appear; (*b*) 1 does not appear; (*c*) neither 0 nor 1 appears; (*d*) at least one of the two digits 0 and 1 does not appear? Let A and B represent the events in (*a*) and (*b*). Express the other events in terms of A and B.

9. If n balls are placed at random into n cells, find the probability that exactly one cell remains empty.

10. At a parking lot there are twelve places arranged in a row. A man observed that there were eight cars parked, and that the four empty places were adjacent to each other (formed *one* run). Given that there are four empty places, is this arrangement surprising (indicative of non-randomness)?

11. A man is given n keys of which only one fits his door. He tries them successively (sampling without replacement). This procedure may require 1, 2, ..., n trials. Show that each of these n outcomes has probability n^{-1}.

12. Suppose that each of n sticks is broken into one long and one short part. The $2n$ parts are arranged into n pairs from which new sticks are formed. Find the probability (a) that the parts will be joined in the original order, (b) that all long parts are paired with short parts.[16]

13. *Testing a statistical hypothesis.* A Cornell professor got a ticket twelve times for illegal overnight parking. All twelve tickets were given either Tuesdays or Thursdays. Find the probability of this event. (Was his renting a garage only for Tuesdays and Thursdays justified?)

14. *Continuation.* Of twelve police tickets none was given on Sunday. Is this evidence that no tickets are given on Sundays?

15. A box contains ninety good and ten defective screws. If ten screws are used, what is the probability that none is defective?

16. From the population of five symbols a, b, c, d, e, a sample of size 25 is taken. Find the probability that the sample will contain five symbols of each kind. Check the result in tables of random numbers,[17] identifying the digits 0 and 1 with a, the digits 2 and 3 with b, etc.

17. If n men, among whom are A and B, stand in a row, what is the probability that there will be exactly r men between A and B? If they stand in a ring instead of in a row, show that the probability is independent of r and hence $1/(n-1)$. (In the circular arrangement consider only the arc leading from A to B in the positive direction.)

18. What is the probability that two throws with three dice each will show the same configuration if (a) the dice are distinguishable, (b) they are not?

19. Show that it is more probable to get at least one ace with four dice than at least one double ace in 24 throws of two dice. (The answer is known as de Méré's paradox. Chevalier de Méré, a gambler, thought that the two probabilities ought to be equal and blamed mathematics for his losses.)

20. From a population of n elements a sample of size r is taken. Find the probability that none of N prescribed elements will be included in the sample,

---

[16] When cells are exposed to harmful radiation, some chromosomes break and play the role of our "sticks." The "long" side is the one containing the so-called centromere. If two "long" or two "short" parts unite, the cell dies. See D. G. Catcheside, The effect of X-ray dosage upon the frequency of induced structural changes in the chromosomes of *Drosophila Melanogaster*, *Journal of Genetics*, vol. 36 (1938), pp. 307–320.

[17] They are occasionally extraordinarily obliging: see J. A. Greenwood and E. E. Stuart, Review of Dr. Feller's critique, *Journal for Parapsychology*, vol. 4 (1940), pp. 298–319, in particular p. 306.

assuming the sampling to be (*a*) without, (*b*) with replacement. Compare the numerical values for the two methods when (i) $n = 100$, $r = N = 3$, and (ii) $n = 100$, $r = N = 10$.

21. *Spread of rumors.* In a town of $n + 1$ inhabitants, a person tells a rumor to a second person, who in turn repeats it to a third person, etc. At each step the recipient of the rumor is chosen at random from the n people available. Find the probability that the rumor will be told r times without: (*a*) returning to the originator, (*b*) being repeated to any person. Do the same problem when at each step the rumor is told by one person to a gathering of N randomly chosen people. (The first question is the special case $N = 1$.)

22. *Chain letters.* In a population of $n + 1$ people a man, the "progenitor," sends out letters to two persons, the "first generation." These repeat the performance and, generally, each member of the rth generation sends out letters to two persons chosen at random. Find the probability that the generations number 1, 2, ..., r will not include the progenitor. Find the median of the distribution, supposing n to be large.

23. *A familiar problem.* In a certain family four girls take turns at washing dishes. Out of a total of four breakages, three were caused by the youngest girl, and she was thereafter called clumsy. Was she justified in attributing the frequency of her breakages to chance? Discuss the connection with random placements of balls.

24. What is the probability that (*a*) the birthdays of twelve people will fall in twelve different calendar months (assume equal probabilities for the twelve months), (*b*) the birthdays of six people will fall in exactly two calendar months?

25. Given thirty people, find the probability that among the twelve months there are six containing two birthdays and six containing three.

26. A closet contains n pairs of shoes. If $2r$ shoes are chosen at random (with $2r < n$), what is the probability that there will be (*a*) no complete pair, (*b*) exactly one complete pair, (*c*) exactly two complete pairs among them?

27. A car is parked among N cars in a row, not at either end. On his return the owner finds that exactly r of the N places are still occupied. What is the probability that both neighboring places are empty?

28. A group of $2N$ boys and $2N$ girls is divided into two equal groups. Find the probability p that each group will be equally divided into boys and girls. Estimate p, using Stirling's formula.

29. In bridge, prove that the probability p of West's receiving exactly k aces is the same as the probability that an arbitrary hand of thirteen cards contains exactly k aces. (This is intuitively clear. Note, however, that the two probabilities refer to two different experiments, since in the second case thirteen cards are chosen at random and in the first case all 52 are distributed.)

30. The probability that in a bridge game East receives m and South n spades is the same as the probability that of two hands of thirteen cards each, drawn at random from a deck of bridge cards, the first contains m and the second n spades.

31. What is the probability that the bridge hands of North and South together contain exactly k aces, where $k = 0, 1, 2, 3, 4$?

32. Let a, b, c, d be four non-negative integers such that $a + b + c + d = 13$. Find the probability $p(a, b, c, d)$ that in a bridge game the players North, East,

South, West have a, b, c, d spades, respectively. Formulate a scheme of placing red and black balls into cells that contains the problem as a special case.

33. Using the result of problem 32, find the probability that some player receives a, another b, a third c, and the last d spades if (*a*) $a = 5, b = 4, c = 3, d = 1$; (*b*) $a = b = c = 4, d = 1$; (*c*) $a = b = 4, c = 3, d = 2$.

Note that the three cases are essentially different.

34. Let a, b, c, d be integers with $a + b + c + d = 13$. Find the probability $q(a, b, c, d)$ that a hand at bridge will consist of a spades, b hearts, c diamonds, and d clubs and show that the problem does *not* reduce to one of placing, at random, thirteen balls into four cells. Why?

35. *Distribution of aces among r bridge cards.* Calculate the probabilities $p_0(r)$, $p_1(r)$, ..., $p_4(r)$ that among r bridge cards drawn at random there are 0, 1, ..., 4 aces, respectively. Verify that $p_0(r) = p_4(52 - r)$.

36. *Continuation: waiting times.* If the cards are drawn one by one, find the probabilities $f_1(r)$, ..., $f_4(r)$ that the first, ..., fourth ace turns up at the rth trial. *Guess* at the *medians* of the waiting times for the first, ..., fourth ace and then calculate them.

37. Find the probability that each of two hands contains exactly k aces if the two hands are composed of r bridge cards each, and are drawn (*a*) from the same deck, (*b*) from two decks. Show that when $r = 13$ the probability in part (*a*) is the probability that two preassigned bridge players receive exactly k aces each.

38. *Misprints.* Each page of a book contains N symbols, possibly misprints. The book contains $n = 500$ pages and $r = 50$ misprints. Show that (*a*) the probability that pages number 1, 2, ..., n contain, respectively, $r_1, r_2, \ldots, r_n$ misprints equals

$$\binom{N}{r_1}\binom{N}{r_2}\cdots\binom{N}{r_n} \div \binom{nN}{r};$$

(*b*) for large N this probability may be approximated by (5.5). Conclude that *the r misprints are distributed in the n pages approximately in accordance with a random distribution of r balls in n cells.* (*Note.* This may be restated as a general limiting property of Fermi-Dirac statistics. Cf. section 5.)

Note: *The following problems refer to the material of section* 5.

39. If r_1 indistinguishable things of one kind and r_2 indistinguishable things of a second kind are placed into n cells, find the number of distinguishable arrangements.

40. If r_1 dice and r_2 coins are thrown, how many results can be distinguished?

41. In how many different distinguishable ways can r_1 white, r_2 black, and r_3 red balls be arranged?

42. Find the probability that in a random arrangement of 52 bridge cards no two aces are adjacent.

43. *Elevator.* In the example (3.*c*) the elevator starts with seven passengers and stops at ten floors. The various arrangements of discharge may be denoted by symbols like (3, 2, 2), to be interpreted as the event that three passengers leave together at a certain floor, two other passengers at another

floor, and the last two at still another floor. Find the probabilities of the fifteen possible arrangements ranging from (7) to (1, 1, 1, 1, 1, 1, 1).

44. *Birthdays.* Find the probabilities for the various configurations of the birthdays of 22 people.

45. Find the probability for a *poker* hand to be a (*a*) royal flush (ten, jack, queen, king, ace in a single suit); (*b*) four of a kind (four cards of equal face values); (*c*) full house (one pair and one triple of cards with equal face values); (*d*) straight (five cards in sequence regardless of suit); (*e*) three of a kind (three equal face values plus two extra cards); (*f*) two pairs (two pairs of equal face values plus one other card); (*g*) one pair (one pair of equal face values plus three different cards).

11. PROBLEMS AND COMPLEMENTS OF A THEORETICAL CHARACTER

1. A population of n elements includes np red ones and nq black ones ($p + q = 1$). A random sample of size r is taken with replacement. Show that the probability of its including exactly k red elements is

$$\binom{r}{k} p^k q^{r-k}. \tag{11.1}$$

2. *A limit theorem for the hypergeometric distribution.* If n is large and $n_1/n = p$, then the probability q_k given by (6.1) and (6.2) is close to (11.1). More precisely,

$$\binom{r}{k}\left(p - \frac{k}{n}\right)^k \left(q - \frac{r-k}{n}\right)^{r-k} < q_k < \binom{r}{k} p^k q^{r-k} \left(1 - \frac{r}{n}\right)^{-r} \tag{11.2}$$

A comparison of this and the preceding problem shows: *For large populations there is practically no difference between sampling with or without replacement.*

3. A random sample of size r *without replacement* is taken from a population of n elements. The probability u_r that N given elements will all be included in the sample is

$$u_r = \binom{n-N}{r-N} \div \binom{n}{r}. \tag{11.3}$$

(The corresponding formula for sampling *with replacement* is given by (11.10) and cannot be derived by a direct argument. For an alternative form of (11.3) cf. problem IV, 9.)

4. *Limiting form.* If $n \to \infty$ and $r \to \infty$ so that $r/n \to p$, then $u_r \to p^N$ (cf. problem 13).

Note: *Problems* 5–13 *refer to the classical occupancy problem* (*Maxwell-Boltzmann statistics*): *That is, r balls are distributed among n cells and each of the* n^r *possible distributions has probability* n^{-r}.[18]

[18] Problems 5–19 play a role in quantum statistics, the theory of photographic plates, G-M counters, etc. The formulas are therefore frequently discussed and discovered in the physical literature, usually without a realization of their classical and essentially elementary character. Probably all the problems occur (although in modified form) in the book by Whitworth quoted at the opening of this chapter.

5. The probability p_k that a given cell contains exactly k balls is given by the binomial distribution (4.5). The most probable number is the integer ν such that $(r - n + 1)/n < \nu \leq (r + 1)/n$. (In other words, it is asserted that $p_0 < p_1 < \ldots < p_{\nu-1} \leq p_\nu > p_{\nu+1} > \ldots > p_r$; cf. problem 15.)

6. *Limiting form.* If $n \to \infty$ and $r \to \infty$ so that the average number $\lambda = r/n$ of balls per cell remains constant, then

$$p_k \to e^{-\lambda}\lambda^k/k!. \tag{11.4}$$

This is the *Poisson distribution,* discussed in chapter VI; see problem 16.

7. Let $A(r, n)$ be the number of distributions leaving *none of the n cells empty.* Show by a combinatorial argument that

$$A(r, n+1) = \sum_{k=1}^{r} \binom{r}{k} A(r-k, n). \tag{11.5}$$

Conclude that

$$A(r, n) = \sum_{\nu=0}^{n} (-1)^\nu \binom{n}{\nu} (n - \nu)^r. \tag{11.6}$$

Hint: Use induction; assume (11.6) to hold and express $A(r-k, n)$ in (11.5) accordingly. Change the order of summation and use the binomial formula to express $A(r, n+1)$ as the difference of two simple sums. Replace in the second sum $\nu + 1$ by a new index of summation and use (8.6).

Note: *Formula* (11.6) *provides a theoretical solution to an old problem but obviously it would be a thankless task to use it for the calculation of the probability x, say, that in a village of* $r = 1900$ *people every day of the year is a birthday. In chapter IV, section* 2, *we shall derive* (11.6) *by another method and obtain a simple approximation formula (showing, e.g., that* $x = 0.135$, *approximately).*

8. Show that *the number of distributions leaving exactly m cells empty is*

$$E_m(r, n) = \binom{n}{m} A(r, n-m) = \binom{n}{m} \sum_{\nu=0}^{n-m} (-1)^\nu \binom{n-m}{\nu} (n - m - \nu)^r. \tag{11.7}$$

9. Show without using the preceding results that *the probability*

$$p_m(r, n) = n^{-r} E_m(r, n)$$

of finding exactly m cells empty satisfies

$$p_m(r+1, n) = p_m(r, n) \frac{n - m}{n} + p_{m-1}(r, n) \frac{m - 1}{n}. \tag{11.8}$$

10. Using the results of problems 7 and 8, show by direct calculation that (11.8) holds. Show that this method provides *a new derivation (by induction on r)* of (11.6).

11. From (11.6) and problem 8 conclude that *the probability of finding m or more cells empty is*

$$\binom{n}{m} \sum_{\nu=0}^{n-m} (-1)^\nu \binom{n-m}{\nu} \left(1 - \frac{m + \nu}{n}\right)^r \frac{m}{m + \nu}. \tag{11.9}$$

(For $m \geqq n$ this expression reduces to zero, as is proper.)

12. *The probability that each of N given cells is occupied is*

$$u(r, n) = n^{-r} \sum_{k=0}^{r} \binom{r}{k} A(k, N)(n - N)^{r-k} \tag{11.10}$$

Conclude that

$$u(r, n) = \sum_{\nu=0}^{N} (-1)^{\nu} \binom{N}{\nu} \left(1 - \frac{\nu}{n}\right)^{r}. \tag{11.11}$$

(Use the binomial theorem. For $N = n$ we have $u(r, n) = n^{-r} A(r, n)$. Note that (11.11) is the analogue of (11.3) for *sampling with replacement.*[19] For an alternative derivation see problem IV, 8.)

13. *Limiting form.* For the passage to the limit described in problem 4 one has $u(r, n) \to (1 - e^{-\rho})^{N}$.

Note: *In problems* 14–19 *r and n have the same meaning as above, but we assume that the balls are indistinguishable and that all distinguishable arrangements have equal probabilities (Bose-Einstein statistics).*

14. The probability that a given cell contains exactly k balls is

$$q_k = \binom{n + r - k - 2}{r - k} \div \binom{n + r - 1}{r}. \tag{11.12}$$

15. Show that when $n > 2$ zero is the most probable number of balls in any specified cell, or more precisely, $q_0 > q_1 > \ldots$ (cf. problem 5).

16. *Limit theorem.* Let $n \to \infty$ and $r \to \infty$, so that the average number of particles per cell, r/n, tends to λ. Then

$$q_k \to \frac{\lambda^k}{(1 + \lambda)^{k+1}}. \tag{11.13}$$

(The right side is known as the *geometric distribution.*)

17. The probability that exactly m cells remain empty is

$$\rho_m = \binom{n}{m} \binom{r - 1}{n - m - 1} \div \binom{n + r - 1}{r}. \tag{11.14}$$

---

[19] Note that $u(r, n)$ may be interpreted as the probability that the *waiting time* up to the moment when the Nth element joins the sample is less than r. The result may be applied to *random sampling digits:* here $u(r, 10) - u(r - 1, 10)$ is the probability that a sequence of r elements must be observed to include the complete set of all ten digits. This can be used as a test of randomness. R. E. Greenwood (Coupon collector's test for random digits, *Mathematical Tables and Other Aids to Computation*, vol. 9 (1955), pp. 1–5) has tabulated the distribution and compared it to actual counts for the corresponding waiting times for the first 2035 decimals of π and the first 2486 decimals of e. The median of the waiting time for a complete set of all ten digits is 27. The probability that this waiting time exceeds 50 is greater than 0.05, and the probability of the waiting time exceeding 75 is about 0.0037.

18. The probability that a group of m prescribed cells contains a total of exactly j balls is

$$(11.15)\quad q_j(m) = \binom{m+j-1}{m-1}\binom{n-m+r-j-1}{r-j} \div \binom{n+r-1}{r}.$$

19. *Limiting form.* For the passage to the limit of problem 4 we have

$$(11.16)\quad q_j(m) \to \binom{m+j-1}{m-1}\frac{p^j}{(1+p)^{m+j}}.$$

(The right side is a special case of the *negative binomial distribution* to be introduced in chapter VI.)

Theorems on Runs. *In problems* 20–25 *we consider arrangements of* r_1 *alphas and* r_2 *betas and assume that all arrangements are equally probable [see example* (4.*d*)*]. This group of problems refers to section* 5*a.*

20. The probability that the arrangement contains exactly k runs of either kind is

$$(11.17)\quad P_{2\nu} = 2\binom{r_1-1}{\nu-1}\binom{r_2-1}{\nu-1} \div \binom{r_1+r_2}{r_1}$$

when $k = 2\nu$ is even, and

$$(11.18)\quad P_{2\nu+1} = \left\{\binom{r_1-1}{\nu}\binom{r_2-1}{\nu-1} + \binom{r_1-1}{\nu-1}\binom{r_2-1}{\nu}\right\} \div \binom{r_1+r_2}{r_1}$$

when $k = 2\nu + 1$ is odd.

21. *Continuation.* Conclude that the most probable number of runs is an integer k such that $\dfrac{2r_1r_2}{r_1+r_2} < k < \dfrac{2r_1r_2}{r_1+r_2} + 3$. (*Hint*: Consider the ratios $P_{2\nu+2} \div P_{2\nu}$ and $P_{2\nu+1} \div P_{2\nu-1}$.)

22. The probability that the arrangement starts with an alpha run of length $\nu \geq 0$ is $(r_1)_\nu r_2 \div (r_1+r_2)_{\nu+1}$. (*Hint*: Choose the ν alphas and the beta which must follow it.) What does the theorem imply for $\nu = 0$?

23. The probability of having exactly k runs of alphas is

$$(11.19)\quad \pi_k = \binom{r_1-1}{k-1}\binom{r_2+1}{k} \div \binom{r_1+r_2}{r_1}.$$

Hint: This follows easily from the second part of the lemma of section 5. Alternatively, equation (11.19) may be derived from (11.17) and (11.18), but this procedure is more laborious.

24. The probability that the nth alpha is preceded by exactly m betas is

$$(11.20)\quad \binom{r_1+r_2-n-m}{r_2-m}\binom{m+n-1}{m} \div \binom{r_1+r_2}{r_1}.$$

25. The probability for the alphas to be arranged in k runs of which k_1 are of length 1, k_2 of length 2, ..., k_ν of length ν (with $k_1 + \ldots + k_\nu = k$) is

$$(11.21)\quad \frac{k!}{k_1!k_2!\ldots k_\nu!}\binom{r_2+1}{k} \div \binom{r_1+r_2}{r_1}.$$

12. PROBLEMS AND IDENTITIES INVOLVING BINOMIAL COEFFICIENTS

1. For integral $n \geq 2$

$$1 - \binom{n}{1} + \binom{n}{2} - + \ldots = 0$$

$$\binom{n}{1} + 2\binom{n}{2} + 3\binom{n}{3} + \ldots = n2^{n-1}$$

(12.1)

$$\binom{n}{1} - 2\binom{n}{2} + 3\binom{n}{3} - + \ldots = 0,$$

$$2 \cdot 1\binom{n}{2} + 3 \cdot 2\binom{n}{3} + 4 \cdot 3\binom{n}{4} + \ldots = n(n-1)2^{n-2}$$

(*Hint:* Use the binomial formula.)

2. Prove that for positive integers n, k

$$\binom{n}{0}\binom{n}{k} - \binom{n}{1}\binom{n-1}{k-1} + \binom{n}{2}\binom{n-2}{k-2} \cdots \pm \binom{n}{k}\binom{n-k}{0} = 0. \tag{12.2}$$

More generally [20]

$$\sum \binom{n}{\nu}\binom{n-\nu}{k-\nu} t^{\nu} = \binom{n}{k}(1+t)^k. \tag{12.3}$$

3. For any $a > 0$

$$\binom{-a}{k} = (-1)^k \binom{a+k-1}{k}. \tag{12.4}$$

If a is an integer, this can be proved also by differentiation of the geometric series $\Sigma x^k = (1-x)^{-1}$.

4. Prove that

$$\binom{2n}{n} 2^{-2n} = (-1)^n \binom{-\frac{1}{2}}{n}. \tag{12.5}$$

5. For integral non-negative n and r and all real a

$$\sum_{\nu=0}^{n} \binom{a-\nu}{r} = \binom{a+1}{r+1} - \binom{a-n}{r+1}. \tag{12.6}$$

(*Hint:* Use equation (8.6). The special case $n = a$ is frequently used.)

6. For arbitrary a and integral $n \geq 0$

$$\sum_{\nu=0}^{n} (-1)^{\nu} \binom{a}{\nu} = (-1)^n \binom{a-1}{n}. \tag{12.7}$$

[*Hint:* Use equation (8.6).]

---

[20] The reader is reminded of the convention (8.5): if ν runs through *all* integers, only finitely many terms in the sum in (12.3) are different from zero.

7. For positive integers r, k

$$\sum_{\nu=0}^{r} \binom{\nu+k-1}{k-1} = \binom{r+k}{k}. \tag{12.8}$$

(*a*) Prove this using (8.6). (*b*) Show that (12.8) is a special case of (12.7). (*c*) Show by an inductive argument that (12.8) leads to a new proof of the first part of the lemma of section 5.

8. In section 6 we remarked that the terms of the hypergeometric distribution should add to unity. This amounts to saying that for any positive integers a, b, n,

$$\binom{a}{0}\binom{b}{n} + \binom{a}{1}\binom{b}{n-1} + \ldots + \binom{a}{n}\binom{b}{0} = \binom{a+b}{n}. \tag{12.9}$$

Prove this by induction. (*Hint*: Prove first that equation (12.9) holds for $a = 1$ and all b.)

9. *Continuation.* By a comparison of the coefficients of t^n on both sides of

$$(1+t)^a(1+t)^b = (1+t)^{a+b} \tag{12.10}$$

prove more generally that (12.9) is true for arbitrary numbers a, b (and integral n).

10. Using equation (12.9), prove that

$$\binom{n}{0}^2 + \binom{n}{1}^2 + \binom{n}{2}^2 + \ldots + \binom{n}{n}^2 = \binom{2n}{n}. \tag{12.11}$$

11. Using equation (12.11), prove that

$$\sum_{\nu=0}^{n} \frac{(2n)!}{(\nu!)^2(n-\nu)!^2} = \binom{2n}{n}^2. \tag{12.12}$$

12. Prove that for integers $0 < a < b$

$$\sum_{k=1}^{a} (-1)^{a-k} \binom{a}{k}\binom{b+k}{b+1} = \binom{b}{a-1}. \tag{12.13}$$

Hint: Using (12.4) show that (12.11) is a special case of (12.9). Alternatively, compare coefficients of t^{a-1} in $(1-t)^a(1-t)^{-b-2} = (1-t)^{a-b-2}$.

13. By specialization derive from (12.9) the identities

$$\binom{a}{k} - \binom{a}{k-1} + - \ldots \mp \binom{a}{1} \pm 1 = \binom{a-1}{k} \tag{12.14}$$

and

$$\sum_{\nu} (-1)^{\nu} \binom{a}{\nu}\binom{n-\nu}{r} = \binom{n-a}{n-r}, \tag{12.15}$$

valid if k, n, and r are positive integers. [*Hint:* Use (12.4).]

14. Using equation (12.9), prove that

$$\sum_{j=0}^{k} \binom{a+k-j-1}{k-j}\binom{b+j-1}{j} = \binom{a+b+k-1}{k}. \tag{12.16}$$

(*Hint:* Apply equation (12.4) back and forth.) Note the important special cases $b = 1, 2$.

15. Referring to the problems of section 11, notice that equations (11.12), (11.14), (11,15), and (11.16) define probabilities. In each the quantities should therefore add to unity. Show that this is implied, respectively, by (12.8), (12.9), (12.16), and the binomial theorem.

16. From the definition of $A(r, n)$ in problem 7 of section 11 it follows that $A(r, n) = 0$ if $r < n$ and $A(n, n) = n!$. In other words

$$\sum_{k=0}^{n} (-1)^{n-k} \binom{n}{k} k^r = \begin{cases} 0 & \text{if } r < n \\ n! & \text{if } r = n. \end{cases} \tag{12.17}$$

(*a*) Prove (12.17) directly by reduction from n to $n - 1$. (*b*) Next prove (12.17) by considering the rth derivative of $(1 - e^t)^n$ at $t = 0$. (*c*) Generalize (12.17) by starting from (11.11) instead of (11.6).

17. If $0 \leq N \leq n$ prove by induction that for each integer $r \geq 0$

$$\sum_{\nu=0}^{N} (-1)^{\nu} \binom{N}{\nu} (n - \nu)_r = \binom{n - N}{r - N} r!. \tag{12.18}$$

(Note that the right-hand member vanishes when $r < N$ and when $r > n$.) Verify (12.18) by considering the rth derivative of $t^{n-N}(t - 1)^N$ at $t = 1$.

18. Prove by induction (using the binomial theorem)

$$\binom{n}{1}\frac{1}{1} - \binom{n}{2}\frac{1}{2} + \ldots + (-1)^{n-1}\binom{n}{n}\frac{1}{n} = 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n}. \tag{12.19}$$

Verify (12.19) by integrating the identity $\sum_{0}^{n-1} (1 - t)^{\nu} = \{1 - (1 - t)^n\}t^{-1}$.

19. Show that for any positive integer m

$$(x + y + z)^m = \sum \frac{m!}{a!b!c!} x^a y^b z^c \tag{12.20}$$

where the summation extends over all non-negative integers a, b, c, such that $a + b + c = m$.

20. Using Stirling's formula, prove that

$$\binom{2n}{n} \sim (\pi n)^{-\frac{1}{2}} 2^{2n}. \tag{12.21}$$

21. Prove that for any positive integers a and b

$$\frac{(a + 1)(a + 2)\cdots(a + n)}{(b + 1)(b + 2)\cdots(b + n)} \sim \frac{b!}{a!} n^{a-b}. \tag{12.22}$$

22. The *gamma function* is defined by

$$\Gamma(x) = \int_0^{\infty} z^{x-1} e^{-z}\, dz \tag{12.23}$$

where $x > 0$. Show that $\Gamma(x) \sim (2\pi)^{\frac{1}{2}} e^{-x} x^{x-\frac{1}{2}}$. (Notice that if $x = n$ is an integer, $\Gamma(n) = (n - 1)!$.)

23. Let a and r be arbitrary positive numbers and n a positive integer. Show that

$$a(a+r)(a+2r)\cdots(a+nr) \sim Cr^{n+1}n^{n+(a/r)+\frac{1}{2}}e^{-n}. \tag{12.24}$$

[The constant C is equal to $(2\pi)^{\frac{1}{2}}/\Gamma(a/r)$.]

24. Using the results of the preceding problem, show that

$$\frac{a(a+r)(a+2r)\cdots(a+nr)}{b(b+r)(b+2r)\cdots(b+nr)} \sim \frac{\Gamma(b/r)}{\Gamma(a/r)}n^{(a-b)/r}. \tag{12.25}$$

25. Prove the following *alternative form of Stirling's formula:*

$$n! \sim (2\pi)^{\frac{1}{2}}(n+\tfrac{1}{2})^{n+\frac{1}{2}}e^{-(n+\frac{1}{2})}. \tag{12.26}$$

26. *Continuation.* Using the method of the text, show that

$$(2\pi)^{\frac{1}{2}}(n+\tfrac{1}{2})^{n+\frac{1}{2}}e^{-(n+\frac{1}{2})-1/24(n+\frac{1}{2})} < n! < (2\pi)^{\frac{1}{2}}(n+\tfrac{1}{2})^{n+\frac{1}{2}}e^{-(n+\frac{1}{2})}. \tag{12.27}$$

27. Extending Stirling's formula, prove that

$$n! \sim (2\pi)^{\frac{1}{2}}n^{n+\frac{1}{2}}\exp\left\{-n+\frac{1}{12n}-\frac{1}{360n^3}+\cdots\right\}. \tag{12.28}$$