**Timestamping**

If you have a printed document and want to prove that it existed on a certain date, you can get it notarized. This is important for copyrighting a document to prove you originated it. This is more difficult with digital data. If there is a date on it can easily be replaced by another. The solution is a timestamping service.

Here is a second scenario. If Alice signs a message, and later decides that she does not want that message to be signed (this is a kind of cheating) then she can anonymously publish her private key and say that anyone could have done the signing. This is called repudiation. So if someone receives a signature from Alice, he or she can demand that Alice use a digital timestamping service. time-stamped. That lessens Alice’s ability to repudiate the signature.

We will give three protocols for timestamping.

Let’s say that Trent is a timestamper. Let $A$ be Alice’s name. Let’s say that Alice’s is the $n$th request for a trusted timestamp that Trent has ever received.

**Timestamping protocol 1.** Alice wants to get a TTS = trusted timestamp on her document. She computes $H$, the hash and sends it to Trent. Trent creates a timestamp $t = \text{time and date when he received } H$. Trent computes $TTS = [(A, H, t, \text{Trent’s address}), d_{\text{Trent}}(\text{mod } n_{\text{Trent}}), \text{Trent’s public key}]$. Technically $t$ is a timestamp. Alice keeps the TTS. Trent stores nothing.

Digistamp does this. Each costs $0.40, they’ve done over a million, none has been called into question.

Problem: Alice could bribe Digistamp to sign with a false $t$.

**Protocol 2.** Alice sends $H$ to Trent. Trent creates $t$ and serial # $n$ (serial # ’s increment by 1 each time). Trent computes $TTS = (\text{hash}(A, H, t, n) d_{\text{Trent}}(\text{mod } n_{\text{Trent}}))$ and sends it to Alice. Every week, Trent publishes last serial number used each day (which Trent signs). Every week, Trent zips collection of week’s TTS’s and signs that and publishes that. Publications are at web site.

PGP does this and also publishes at alt.security.pgp user group. Solves the bribing problem.

Problem: Depends on popularity of Trent for trust (must trust that they’re not messing with old posts on web site or user group). Hard to have small competitors. Lots of storage.

**Protocol 3 (Stuart Haber and W. Scott Stornetta).** Alice sends $H_n$, the hash of her document, to Trent. Note $I_n = A$. Trent computes $TTS_n = (n, t_n, I_n, H_n; L_n) d_{\text{Trent}}(\text{mod } n_{\text{Trent}})$ where $L_n = (t_{n-1}, I_{n-1}, H_{n-1}, H(L_{n-1}))$. Note $L_n$ connects connects the nth with the $n-1$st.

Later Trent sends Alice $I_{n+1}$. In homework you’ll describe storage.

Can Alice or Alice and Trent together change $t_n$ later? David, the doubter, can ask Alice for the name of $I_{n+1}$. David contacts $I_{n+1}$. Then $I_{n+1}$ raises his $TTS_{n+1}^d (\text{mod } n_{\text{Trent}})$. Note, the fifth entry is $L_{n+1} = t_n, \ldots$. So David sees Alice’s timestamp $t_n$ in $I_{n+1}$’s $TTS_{n+1}$. David can also contact $I_{n-1}$ if he wants to. This prevents Alice and Trent from colluding to change $t_n$.


Problem: To check, need to contact people before and after Alice.

The author is not aware of a company implementing Protocol 3.