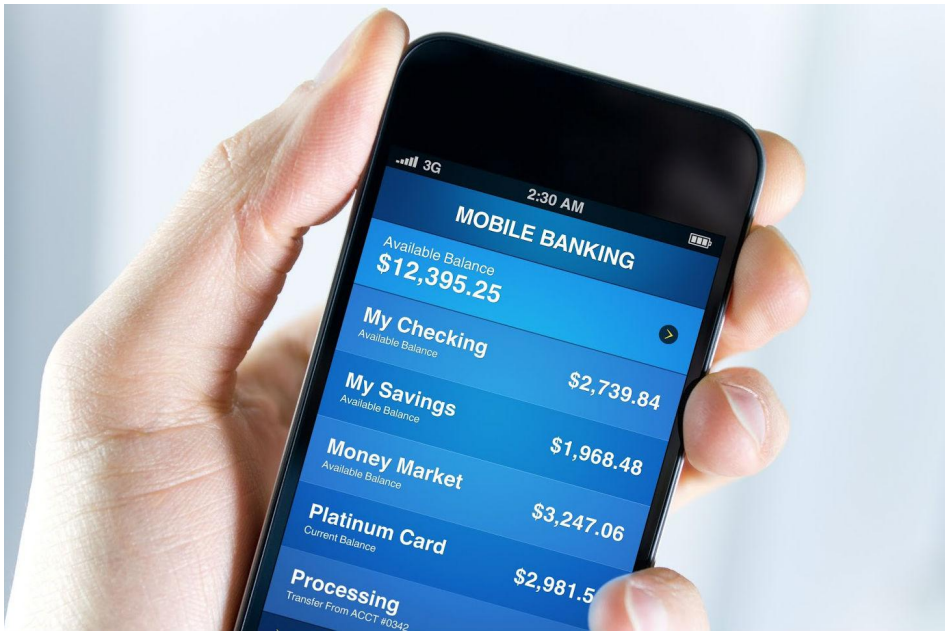
Single-Use-Seals

Peter Todd

July 3rd 2018

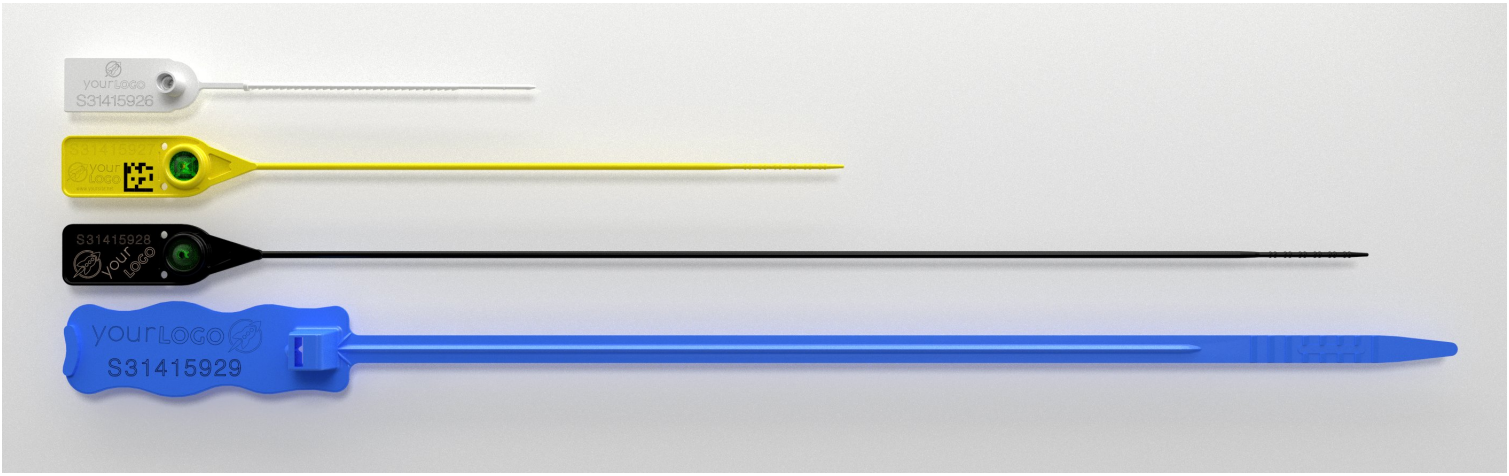
Building on Bitcoin

What problem are we trying to solve?



What problem are we trying to solve?

Physical Single-Use-Seal



Cryptographic Single-Use-Seal

$$\begin{aligned} \text{Gen}(p) &\rightarrow I \\ \text{Close}(I, m, s) &\rightarrow w_I \\ \text{Verify}(I, w_I, m) &\rightarrow \text{bool} \end{aligned} \tag{1}$$

Single-Use-Seal Guarantee

Secure if $\nexists\ m_1, w_1, m_2, w_2$ where $m_1 \neq m_2$ such that Verify(I, w_1, m_1) and Verify(I, w_2, m_2) return true.

Blockchain Implemented Via Single-Use-Seals

Series of blocks $B_i = (w_i, l_i, b_i)$ such that
$\text{Verify}(l_i, w_{i+1}, H(l_i|b_i))$ is true for each block.

Implementation: Seal Structure

```
24 #[derive(Debug,Clone)]
25 pub struct Seal {
26     pub txid: Sha256dHash,
27     pub idx: u32,
28 }
```

Implementation: Seal Serialization

```
55 impl<S: SimpleEncoder> ConsensusEncodable<S> for Seal {
56     fn consensus_encode(&self, e: &mut S) -> Result<(), S::Error> {
57         self.txid.consensus_encode(e)?;
58         e.emit_u32(self.idx)
59     }
60 }
61 impl<D: SimpleDecoder> ConsensusDecodable<D> for Seal {
62     fn consensus_decode(d: &mut D) -> Result<Self, D::Error> {
63         Ok(Seal{
64             txid: Sha256dHash::consensus_decode(d)?,
65             idx: d.read_u32()?,
66         })
67     }
68 }
```

Implementation: Witness Structure

```
70 #[derive(Debug,Clone)]
71 pub struct Witness {
72     pub tx: Transaction,
73     pub block_height: u32,
74 }
```

Implementation: Witness Validation

```rust
#[derive(Debug,Clone,PartialEq,Eq)]
pub enum WitnessError {
    MissingInput,
    MissingOutput,
    BadOutput,
}

impl Witness {
    pub fn validate(&self, seal: &Seal, digest: &[u8]) -> Result<(), WitnessError> {
        let expected_script_pubkey = script::Builder::new()
                                            .push_opcode(OP_RETURN)
                                            .push_slice(digest)
                                            .into_script();

        for (i, txin) in self.tx.input.iter().enumerate() {
            if txin.prev_hash == seal.txid &&
               txin.prev_index == seal.idx
            {
                if let Some(txout) = self.tx.output.get(i) {
                    if txout.script_pubkey == expected_script_pubkey {
                        return Ok(());
                    } else {
                        return Err(WitnessError::BadOutput);
                    }
                } else {
                    return Err(WitnessError::MissingOutput);
                }
            }
        }
        Err(WitnessError::MissingInput)
    }
}
```

Implementation: Chain of Seals

```
17 pub struct Target {
18     pub digest: Sha256dHash,
19     pub seal: Seal,
20 }
21
22 pub struct Link {
23     pub witness: Witness,
24     pub target: Target,
25 }
26
27 pub struct Chain {
28     pub genesis: Target,
29     pub links: Vec<Link>,
30 }
```

Scaling: Proof-of-Publication Seal

Seal defined as $I = (B_n, p)$ where B_n is an initial block and p is a pubkey.
Verify takes $w_I = (B_n..B_m)$ and verifies that CheckSig(p, m, B_i) returns false for all $n \leq i < m$, and true for $i = m$.

Scaling: *Indexed Proof-of-Publication Seal*

If block B_n is a merkelized key-value tree, proving $B_n[p] = m$ takes $O(\log_2(m))$ bytes.

PoP Seal Proof Cost

$$\underbrace{1250\text{bytes}}_{\text{tx w/ SPV proof}} + \underbrace{\frac{32\text{bytes}}{\text{level}}32\text{levels}}_{\text{key-value path}} + \underbrace{100\text{bytes}}_{\text{sig}} \approx \frac{3\text{kB}}{\text{block}}$$

$$\frac{356\text{days}}{\text{year}} \times \frac{12\text{blocks}}{\text{day}} \times \frac{3\text{kB}}{\text{block}} \Longrightarrow \frac{13\text{MB}}{\text{seal}\cdot\text{year}}$$

Scaling: Proof-of-Publication Seal

Why isn't this implemented yet?

Thank you!