

## Project Sanxenxo

# Privacy Preserving Smart Metering

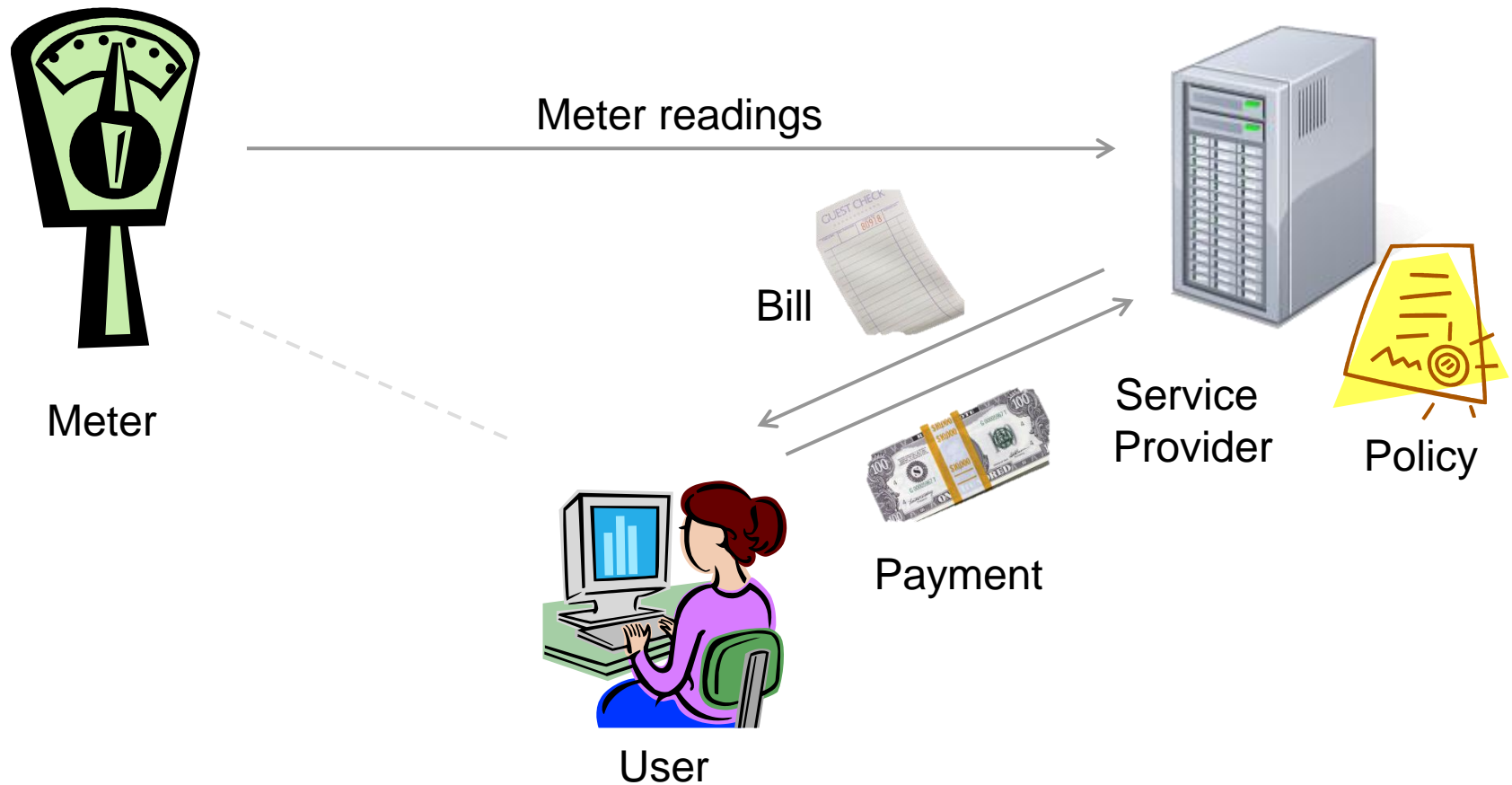
George Danezis

Microsoft Research, Cambridge

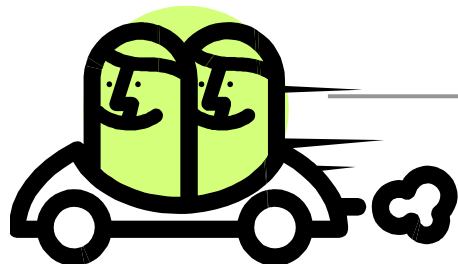
Alfredo Rial

KU Leuven

# The metering setting



# Vehicles



Vehicle

(Location, engine, time)

GPS Location



Insurance Company  
Road Taxation Authority  
Congestion Charging



Payment



User



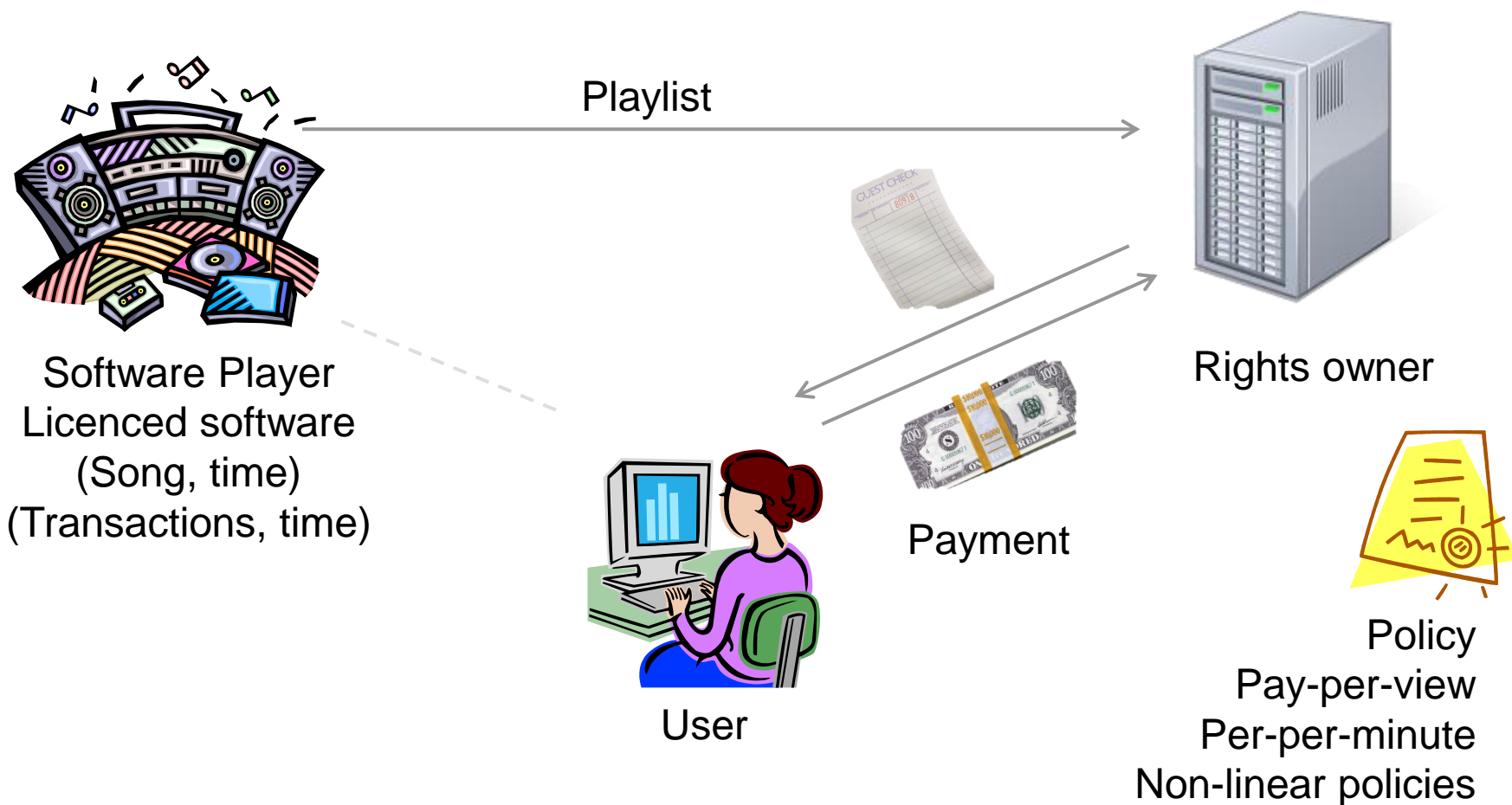
Policy

Rate per mile, given road & time

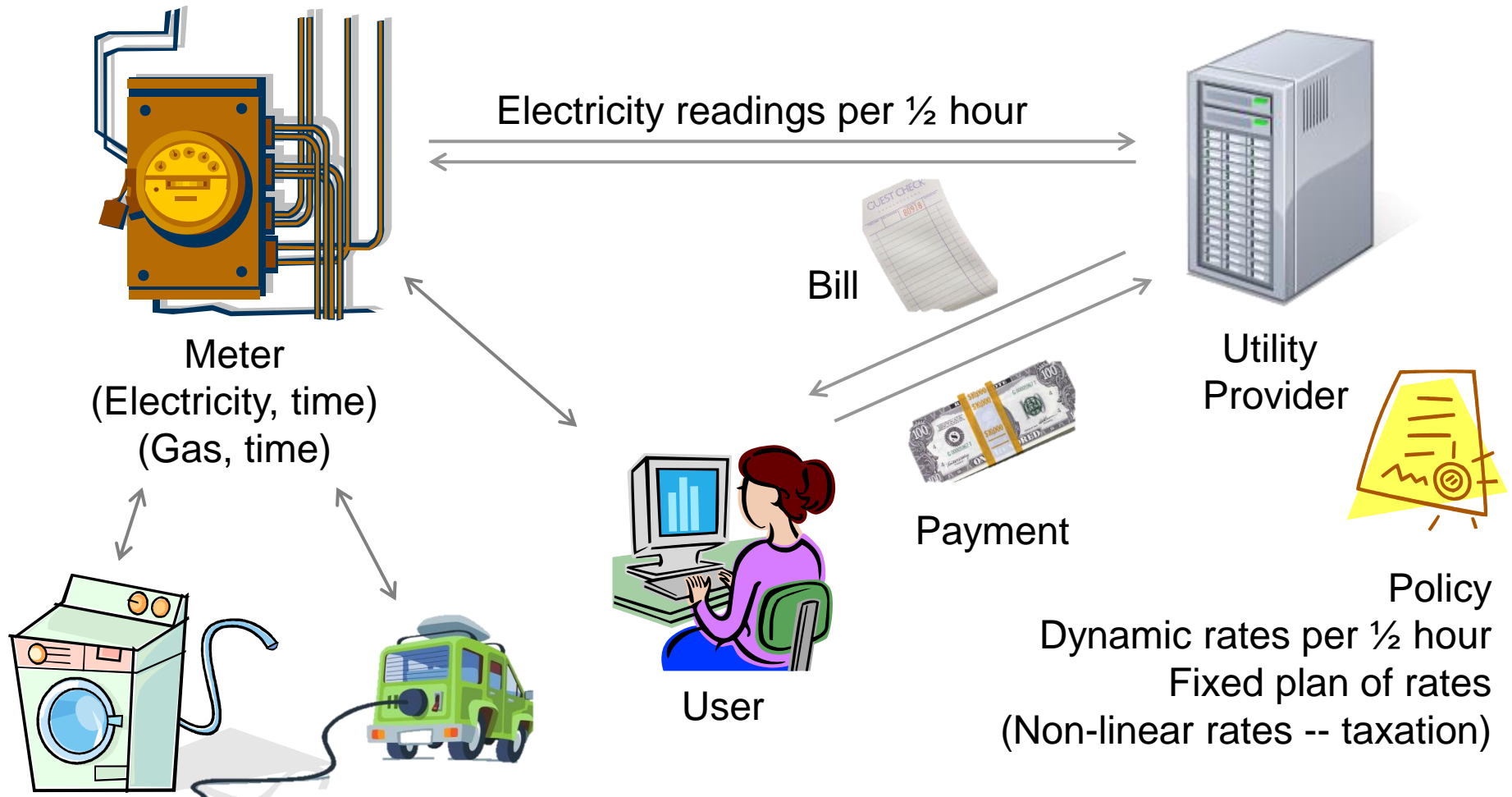
Carmela Troncoso, George Danezis, Eleni Kosta, Bart Preneel: **Pripayd: privacy friendly pay-as-you-drive insurance.** WPES 2007: 99-107

J. Balasch, A. Rial, C. Troncoso, C. Geuens, B. Preneel, and I. Verbauwhede, "PrETP: **Privacy-Preserving Electronic Toll Pricing,**" In *19th USENIX Security Symposium 2010*, Usenix, 16 pages, 2010.

# Software / DRM model



# Utility model



# Smart-grid for electricity

- USA: Energy Independence and Security Act of 2007
  - American Recovery and Reinvestment Act (2009, \$4.5bn)
- EU: Directive 2009/72/EC
- UK: deployment of 47 million smart meters by 2020

BUT:

“The Dutch First Chamber considers the mandatory nature of smart metering as an unacceptable infringement of citizens’ privacy and security”

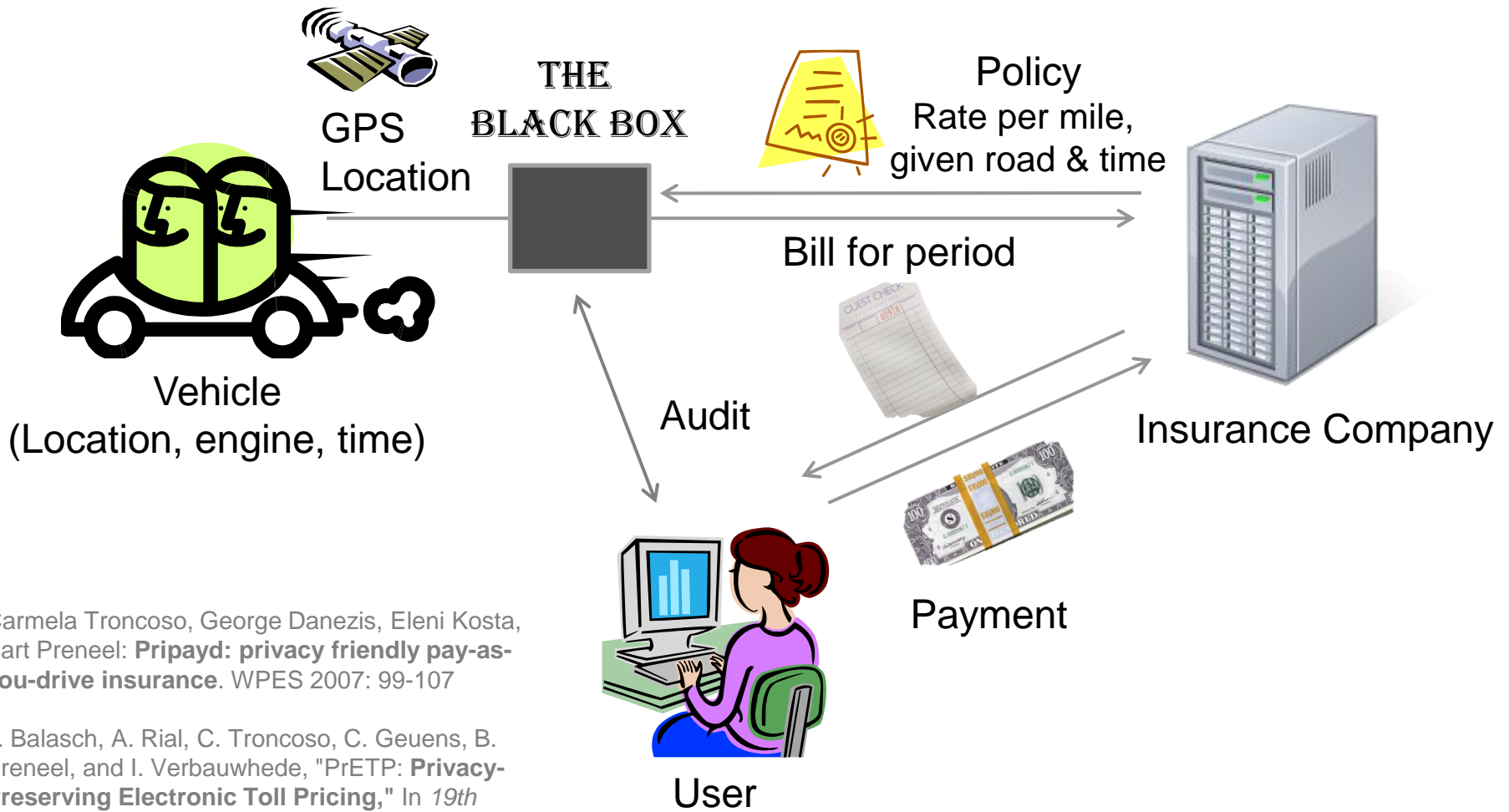
# Privacy issues

- Meter readings are sensitive:
  - Location for vehicles leaks information
    - visiting the hospital often? Late for work again?
  - Content for DRM can cause public embarrassment
    - watching Disney movies? Listening to Celine Dion?
  - Readings for smart-grids reveal lifestyle
    - Were you in last night? You do like watching TV don't you? Another ready meal in the microwave? Has your boyfriend moved in?
- Adding insult to the injury for smart-grids
  - Proposed centralised databases of readings (UK)
  - Mandatory to receive service
  - Ability to switch off / switch to prepaid meters

# Ingredients for a solution

- General form of the metering problem is common amongst all applications.
  - Devil in the detail: abuse control, back channels, ...
- User knows all data: Readings & Policy
  - Personal data & fair contract considerations
- Integrity for others
  - Providers should care about money not PII
- Tariffs can be complex
  - 30p for the first 10miles of motorway, then 15p per mile.
- Agility is important
  - Change policies without modifying infrastructure

# Simple approach: PriPAYD



Carmela Troncoso, George Danezis, Eleni Kosta, Bart Preneel: **PriPAYD: privacy friendly pay-as-you-drive insurance**. WPES 2007: 99-107

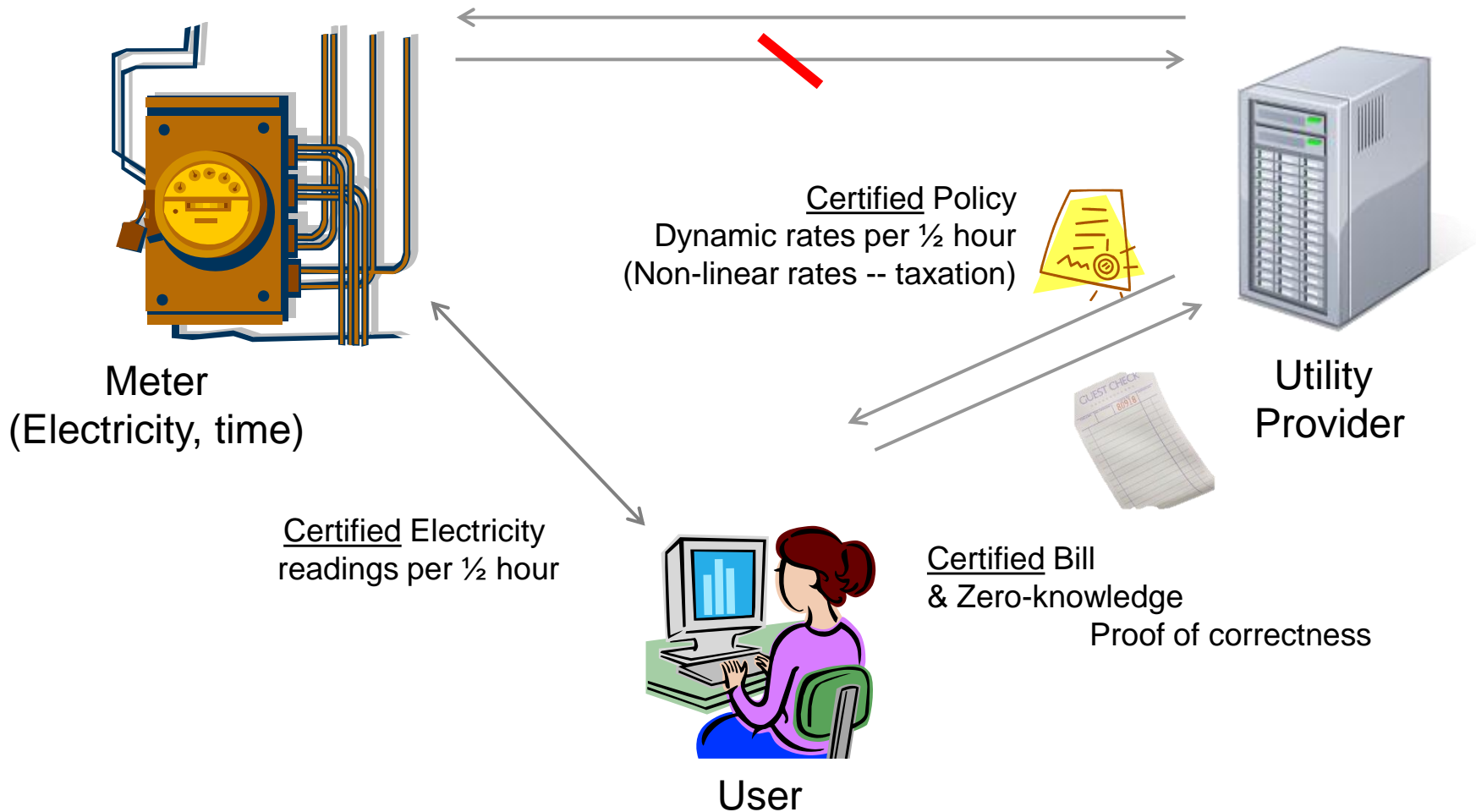
J. Balasch, A. Rial, C. Troncoso, C. Geuens, B. Preneel, and I. Verbauwhede, "PrETP: **Privacy-Preserving Electronic Toll Pricing**," In *19th USENIX Security Symposium 2010*, Usenix, 16 pages, 2010.

# Problems with trivial approach

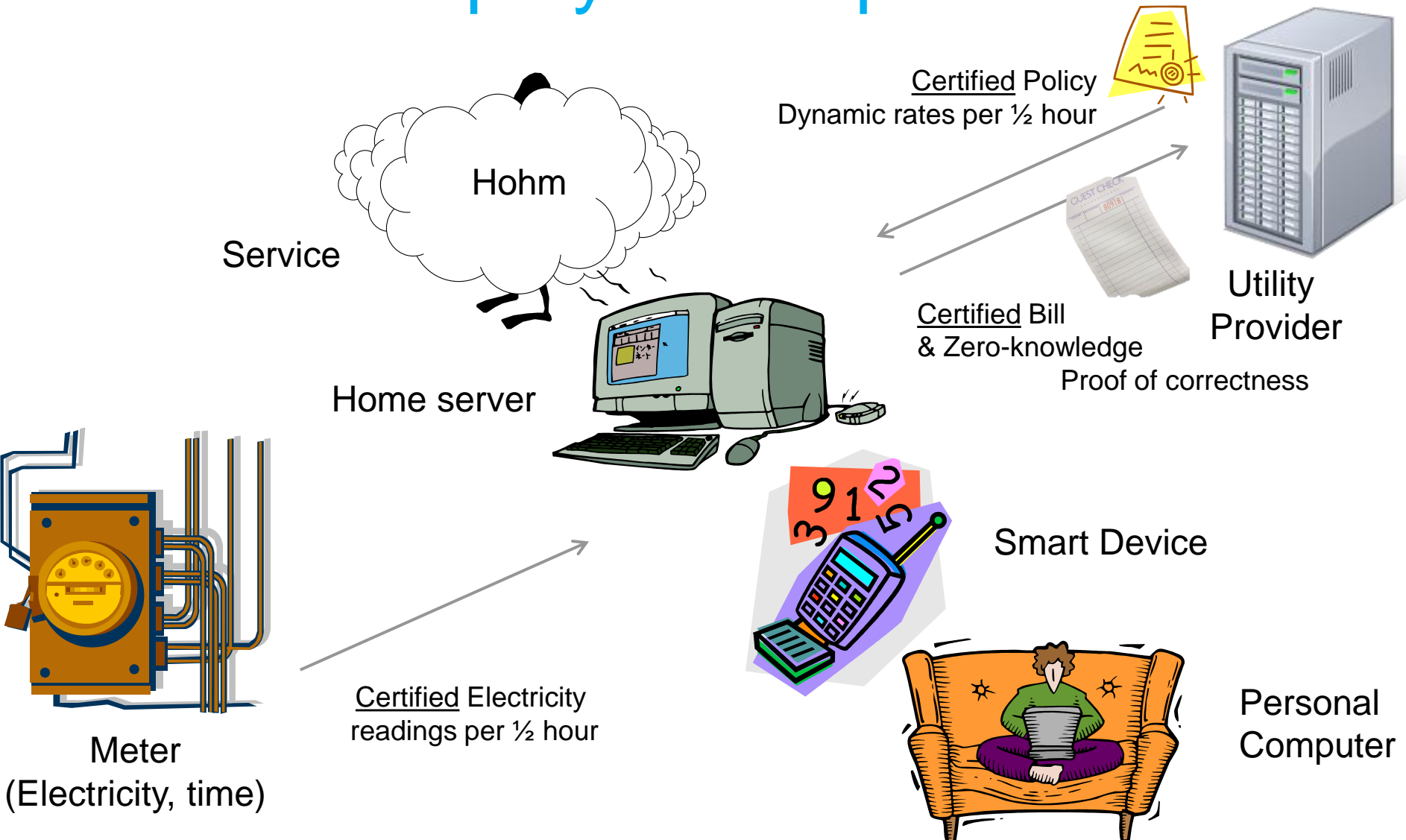
- Approach: The meter computes final bills
- Problems:
  - **Agility**: re-program meter to execute new policy in TCB.
  - **Certification**: meters are certified by national authorities, separate from utilities. Now they need to run utility code.
  - **Incentive misalignment**: the party that should not receive the data commissions & maintains the security system.
  - **Re-use**: use the same data for road insurance & congestion charging – 2 programs, different principals.
  - **End-to-end verification**: how do you know this is the correct bill when it reaches the utility – manual inspection of secure black boxes.
- Better than no privacy!

# Our approach

(A) Certified readings & policy (B) ZK proof of bill & verification



# Deployment options



# The gory details

- 4 stages: setup, initialize, consume, pay
- Generic protocol:
  - Supports any tariff policy that can be expressed as table look-ups and polynomial splines.
- Fast Billing protocol:
  - Special case: policy is public, and selection of rate independent of reading.
  - Very fast.
  - No really ... as fast as calculating the bill without fancy crypto.

# Setup Phase



User

Compute key pair  
 $(sk_U, pk_U) \leftarrow UKeyGen(1^k)$



Compute key pair  
 $(sk_M, pk_M) \leftarrow MKeyGen(1^k)$



Compute key pair  
 $(sk_P, pk_P) \leftarrow PKeyGen(1^k)$

Compute commitment params  
 $par \leftarrow ComSetup(1^k)$

# Initialization Phase



User

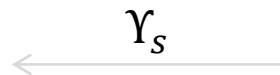


Choose pricing policy  $\Upsilon: (cons, other) \rightarrow price$

Signs pricing policy

$\{\sigma \leftarrow PSign(sk_p, \langle cons, other, price \rangle)\}$

$\Upsilon_s$



$\Upsilon_s = \{\sigma\}$

Verify signed policy

$\{0,1\} \leftarrow PVerifySig(pk_p, \sigma, \langle cons, other, price \rangle)$

# Consumption Phase



User

Read ( $cons, other$ )

Commits to consumption and other

$$(C_{cons}, open_{cons}) \leftarrow Commit(par, cons)$$

$$(C_{other}, open_{other}) \leftarrow Commit(par, other)$$

Signs commitments

$$\sigma \leftarrow MSign(sk_m, \langle d_M, C_{cons}, C_{other} \rangle)$$

$$\xrightarrow{(\sigma, d_M, cons, open_{cons}, other, open_{other})}$$

Verifies commitment openings

$$\{0,1\} \leftarrow Open(par, C_{cons}, cons, open_{cons})$$

$$\{0,1\} \leftarrow Open(par, C_{other}, other, open_{other})$$

Verifies signature

$$\{0,1\} \leftarrow MVerifySig(pk_m, \sigma, \langle d_U, C_{cons}, C_{other} \rangle)$$

# Payment Phase (I)



User

For all tuples  $(cons, other)$  output by M

Compute price  $Y: (cons, other) \rightarrow price$

Commit to price  $(C_{price}, open_{price}) \leftarrow Commit(par, price)$

Proof knowledge of signature that bind consumption and price

$$\pi \leftarrow NIPK\{(\sigma, price, open_{price}, cons, open_{cons}, other, open_{other})\}:$$

$$(C_{price}, open_{price}) \leftarrow Commit(price) \wedge$$

$$(C_{cons}, open_{cons}) \leftarrow Commit(cons) \wedge$$

$$(C_{other}, open_{other}) \leftarrow Commit(other) \wedge$$

$$1 \leftarrow PVerifySig(pk_p, \sigma, \langle cons, other, price \rangle)\}$$

# Payment Phase (II)



User



Aggregate prices and openings

$$fee = \sum_{k=1}^N price_k \quad open_{fee} = \sum_{k=1}^N open_{price}$$

Compose a payment message

$$m = (fee, open_{fee}, \{\sigma, \langle d_U, C_{cons}, C_{other} \rangle, C_{price}, \pi\})$$

Sign payment message

$$s_m \leftarrow USign(sk_U, m)$$

$(m, s_m)$



Verify Signature

$$1 \leftarrow UVerifySig(pk_U, s_m, m)$$

Verify proofs  $\pi$

Aggregate commitments to price

$$C_{fee} = \prod_{k=1}^N C_{price_k}$$

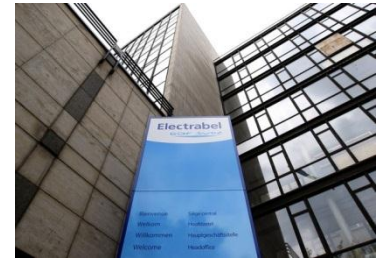
Verify Opening of  $C_{fee}$

$$1 \leftarrow Open(C_{fee}, fee, open_{fee})$$

# Zero-Knowledge lookups?

- You can do them!
  - Certify a table by signing each row using a re-randomizable signature (CL) – each column is one message of the signature.
  - Given a commitment to a key, prove you have a signature with that message in the column.
  - (or key is in secret interval defined by two columns)
- Uses:
  - Table lookups to map location to road type / rate.
  - Used to build arbitrary functions using splines.
- Cost of use: 1 CL signature proof

# The blazing fast protocol



Open Readings

$\{ \dots, (i, cons_i, open_i) \dots \}$

Policy

$\{ \dots, i \rightarrow rate_i, \dots \}_{sign}$



Blind Readings

$\{ \dots i, C_i = g^{cons_i} h^{open_i}, \dots \}_{sign}$



Prove

$Bill = \sum_i rate_i \cdot cons_i$

$Open' = \sum_i rate_i \cdot open_i$



User



Blind readings &  
 $\{ Bill, Open' \}_{sign}$

Verify

(Verify all signatures)

$\prod_i C_i^{rate_i} = g^{Bill} h^{Open'}$

# How do we know it works?

- Proofs & definitions in the UC model
  - Abstract functionality defining metering & billing.
  - Proof that our protocols are indistinguishable from the abstract functionality.
  - Use of lemmas from standards primitives:
    - Commitments, signatures, ZK proofs

# Implementation

- Putting together all the crypto to test speed (and feasibility)
- Generic libraries for zero-knowledge proofs in  $Z_{pq}$ 
  - Proof of knowledge of representation, equality, linear equations, inequality, range, CL signatures, lookup.
  - Fast crypto operations: Montgomery multiplications & pre-computed tables for interleaved exponentiation.
  - C++ abstraction to build custom protocols using ZK proofs.
- No integration into meter or software yet.

# Performance

2 reference platforms (using 1 core):

- 32 Bit Win 7 – Intel Core2 DUO P9600 @ 2.66GHz  
(2 cores) / 4GB Ram (3.49GB Usable)
- 64 Bit Win Server Enterprise – Intel Xeon E5440 @ 2.83GHz  
(4 cores / 2 processors) / 32GB Ram

Reference problem: bill 1000 entries, with lookups (Standard) or flat rate (fast)

	Standard x86	Standard amd64	Fast x86	Fast amd64
Certify policy	21.6877/s	121.187/s	$\epsilon$	$\epsilon$
Calculate entry	66.1169/s	348.782/s	<u>298295/s</u>	<u>199826/s</u>
Prove entry	18.3715/s	101.591/s	$\epsilon$	$\epsilon$
Verify entry	7.47031/s	43.9169/s	34636.5/s	90051.4/s

>90% of time spend in modular multiplication

\* Thanks to MS XCG: Brian LaMacchia, Tolga Acar, Mira, Mira Belenkiy & Dan Shumow

# Conclusion

- Smart metering can be done without leaking all information
- Side information can be revealed (and is certified) for other uses -- fraud detection.
- Tariff structure can change as fast as software can be updated on untrusted machines.
- Homomorphic proof protocols as fast as uncertified calculations.
- General protocols well within realm of real-time.
- Generic library for certifying calculations in zero-knowledge – what can you do with it?