From: Miller, Carl A. (Fed)
To: Brandao, Luis (IntlAssoc)
Subject: Re: Process diagram for the randomness beacon
Date: Tuesday, October 31, 2017 1:05:47 PM

Ok, thanks, would be good to know your thoughts.

Part of the question is whether there's an aid to have a computer-verified proof of security for a particular process. If the proof of security is simple enough that an expert can just see it all at first glance, then we may not need to add anything. But in a more complex system, perhaps computer-verification could help.

-Carl

———

Carl A. Miller
Mathematician, Computer Security Division
National Institute of Standards and Technology
Gaithersburg, MD

From: "Brandao, Luis (IntlAssoc)" <luis.brandao@nist.gov>
Date: Tuesday, October 31, 2017 at 11:20 AM
To: "Miller, Carl A. (Fed)" <carl.miller@nist.gov>
Subject: Re: Process diagram for the randomness beacon

Hi Carl,
When getting a chance, I'll take a look at your paper and then comment back. I actually haven't heard before of about graphical security analysis, so I'm interested in reading about it.
Thanks,
Luís

From: Miller, Carl A. (Fed)
Sent: Thursday, October 26, 2017 12:27:08 PM
To: Brandao, Luis (IntlAssoc); beacon
Subject: Re: Process diagram for the randomness beacon

Hi all –

Luis, thanks for sending out the new version of the process diagram (it's great to have everything in one place – looks nicely arranged). I have ideas for a possible next step to take from here, and I'd be interested in getting comments on it:

We could try to augment some of the security analyses for the beacon with diagrammatic &

computer verified proofs. The goal could be to enhance trust even more by adding some new angles to the security analysis. For example, we're doing a graphical & computer security analysis of quantum random number generation in this paper:

https://arxiv.org/pdf/1705.09213.pdf

This is written for quantum crypto, but the approach can be carried over to general crypto. I was thinking possibly of taking the process diagrams that John & Luis have developed and trying to do some similar proofs. (We might start with the hash-chaining problem that John neatly explained in his slides, since that's a good test case.)

So the question (for people with more experience than me ☺): does it help to add proofs to NIST standards? Is this something the users or the higher-ups tend to find useful? I think I'll pursue this in any case because it's theoretically interesting, but it would be neat if it can give something back to the project also.

-Carl

———

Carl A. Miller
Mathematician, Computer Security Division
National Institute of Standards and Technology
Gaithersburg, MD

From: "Brandao, Luis (IntlAssoc)" <luis.brandao@nist.gov>
Date: Tuesday, October 24, 2017 at 6:22 PM
To: beacon <beacon@nist.gov>
Subject: Re: Process diagram for the randomness beacon

Hello,

As a way of trying to figure out the relations between all elements, I adapted the previous diagram to include symbols for all variables, draw a boundary for the internal engine (leaving the HSM and the time-server outside), correcting some arrows (randomRandomValue and precomitmentValue w.r.t. "current record" vs. "next record"), explain the previous output values, and showing only one time the elements that constitute the pulse. I'm sending an editable svgz and a PDF version. Feel free to edit and/or suggest changes.

Out of convenience of reducing the number of line intersections, I made a little tweak, which was to change (in the drawing) the vertical order of signature and output value. If you think this is bad, I can reverse it back.

-- Luís

From: Brandao, Luis (IntlAssoc)
Sent: Thursday, September 28, 2017 6:16:16 PM
To: Kelsey, John M. (Fed); Miller, Carl A. (Fed); Peralta, Rene (Fed); Booth, Harold (Fed)
Subject: Re: Process diagram for the randomness beacon

Thanks John. Below are some comments/doubts/suggestions.

(Note: don't feel obligated to reply in writing, as I'm taking time to write too much :-) ... we can discuss some of these things at some other meeting.)

======
R1. About Q1 and Q2:

OK, so I'm getting that the goal of mentioning "Time service (remote)" in the diagram is to convey that the "CPU clock" is synchronized with some external global clock. Suggestion: might be clarifying to add "synchronization", to become "Time synchronization service (remote)"; also be clarifying to use "NTP" as a label to the arrow.

Regular NTP is not cryptographic --- it simply synchronizes a client clock based on estimated round-trip delays with a server (or peers). I had asked about crypto timestamps with another intention (related to Q7), which was having the beacon using for its input something signed by a timestamp certification authority, thereby reducing the time available for some brute-force attacks. As you mentioned, this would imply another level of complexity to accommodate an extra input. Yet, I now wonder about how the randomness beacon may be vulnerable to attacks to the NTP protocol (which is already something relying on the outside).

- (i) for the purpose of synchronization, is the beacon interacting with the outside world using a simple NTP protocol without authenticated communication? (I imagine that the beacon generator inside NIST may be using some internal NIST NTP server. For example, by running the command "w32tm/query /peers" on my windows machine I get that my computer synchronizes with peer "WS018.campus.NIST.gov", so the attacker mentioned in question would either have to compromise this NIST server, or would have to be otherwise actively interfering with the internal NIST network.)

- (ii) is it conceivable that an active attacker is able to impersonate (or take over) a NTP server just to make the beacon believe a year has passed without pulses? How would the beacon react in terms of hashing previous pulses if it realizes/interprets that it has not send any pulse for several minutes?

- (iii) what if the result of NTP synchronization would imply (in the beacon's perspective) that time moved backward more than a minute? Would the beacon then generate a new pulse with the same timestamp as a previous pulse? (here the previous question Q8 is pertinent --- what about having a sequential integer counter somewhere?)

On a subsequent search, I meanwhile found that NIST already has an authenticated NTP service :-)
https://www.nist.gov/pml/time-and-frequency-division/time-services/nist-authenticated-ntp-service
Using this service could be good for internal defense from a network attacker trying to interfere with the NTP protocol. This is nonetheless only based on symmetric keys, which does not allow the beacon to use authenticated messages to assure the outside world that the integrity of the timestamps was preserved.

======
R2. Still related with timestamps (including Q5).
What is there in "timestamp" (T_i) besides the values "year" (Ty_i), "month" (Tm_i), "day" (Td_i) and "hour" (Th_i). Is it just the minutes and seconds? Could the minutes contained in "timestamp" (T_i) conceivably ever be different from the independent field "minute" (Tm_i)?

======
R3. About Q3 ("previous pulses").
I think I got it. Suggestion for diagram: the "final pulse" field could be followed with an arrow to the right, labeled as "next record"; it could then have an additional arrow, directed to the bottom, labeled "broadcast" (or something similar). Then, further up in the page, the small grey box "previous pulses" could be replaced by two arrows pointing to the a new SHA512 oval. One arrow would be labeled "previous link" and the next would be labeled "previous chain-link". OUtbounding from this Hash there would be a new arrow pointing to inside the large grey box, to the "previous" field, which could now be renamed as "chain-link". Finally, the "previous" field in the light-grey box in the bottom could also have a right directed arrow sub-labeled as "next iteration".

======
R4. About localRandomValue vs. precommitmentValue.

Maybe I'm getting something wrong, but from my current understanding I think the diagram might be misleading with respect to which localRandomValue vs. precommitmentvalue is used

in each iteration. I'm assuming that an arrow directed to the right margin means something being cached to be used in the next iteration, and that an arrow coming from the left margin of the page means something that was calculated in the previous iteration. If this is correct, then I would suggest:

- The arrow of the "precommitment value" calculated in iteration i (as a hash of the localRandomValue) should be directed to the left (instead of to the right), toward the respective field inside the grey box, because this is really what is going to be used in this iteration when computing the full pulse. Correspondingly, the current arrow coming from the left margin (toward the grey box) should be deleted.

- Conversely, the "localRandomValue" calculated as an hash of two random values should have an outbound arrow to the right margin, rather than to the grey box, because this value will only be used in the clear in the next iteration. Correspondingly, the "localRandomValue" in the grey box should have a corresponding inbound arrow coming from the left margin of the page.

======
R5. About Q4 ("pre-commitment values")
Using your notation A[t] (denoting the pulse at time t), I get why A[t].precommitmentValue (= hash(A[t+1].localRandomValue)) cannot depend on the full A[t+1]. The previous question was however about using other elements of A[t]. For example, A[t].precommitmentValue could conceivably be = Hash(S_{t-1} || A[t+1].localRandomValue), where S_{t-1} is the signature of the previous pulse.

Reflecting a bit more, I was now not able to formulate a concrete attack that would justify this. Nonetheless, if useful, I leave below some reflection about this. (At some points I'm just making very informal arguments, that I still have to rethink.)

The question in the previous email was about preventing advanced generation of precommitmentValue. However, on a new look I would now initially change the question to be about preventing advanced generation of localRandomValue. By "preventing" advanced generation of an element I mean here preventing that it can be done without interaction with the HSM. So the question would now become: could we make h1_i = A[t].localRandomValue = Hash(S_t || r1_t || r2_t), therefore implying that an advance generation of localRandomValues requires the use of the HSM and the knowledge of the previous pulse. In other words, this would imply that the "localRandomValue" would be fresh at each iteration, whereas currently all "localRandomValue"s for the upcoming year could could be generated today (without interaction with the HSM).

(Another technical doubt: do HSMs allow including a sequential number in signatures? If yes,

this could be used to give the world an assurance that, unless the HSM had been tampered with, the HSM had not been used in any brute-force attack.)

Considering the above, the more relevant question would now be: what could a malicious NIST do with advanced knowledge of a year's worth of localRandomValues? Maybe not much, because each "output value" in each pulse stills depend on the previous pulses, which in turn also depends on previous signatures, which require the use of the HSM. However, if a "feeling" of security is only dependent on "trust" that the HSM is not being used in advance, then my next question would be why is it technically relevant to even bother generating local randomness. Specifically, if signature values are unpredictable before generation, signatures would be sufficient to generate a chain of randomly-looking unpredictable values. So I think that part of the "trust" that a brute-force attack (i.e., not to invert hashes, but to bias the output) may in practice also depend on trust that there is a process ensuring that localRandomValues are being generated somewhat freshly. If that's the case, then there might be value in indeed ensuring that localRandomValues on an iteration depend on the signature from the previous interaction.

Overall, the previous paragraph is already depending on a fuzzy argument about trust. Perhaps further technical protection against advanced generation of pulses is only possible by using (either a multi-beacon protocol or) an external service of cryptographic timestamping.

======
R6. About Q6 and Q7
OK. For the purpose only of obtaining protection against the mentioned brute-force attack (i.e., reducing the available time down to about a single minute, rather than arbitrarily long time), I wonder if, in practice, the use of a time-certification authority would be less, equal or more complex than combining another beacon?

--Luís

From: Kelsey, John M. (Fed)
Sent: Wednesday, September 27, 2017 6:44:03 PM
To: Brandao, Luis (IntlAssoc); Miller, Carl A. (Fed); Peralta, Rene (Fed); Booth, Harold (Fed)
Subject: Re: Process diagram for the randomness beacon

Luis,

I'll give you my best answers, but I don't promise to be right....

Q1. What's the difference between the CPU clock time (t_i) and the "timestamp" T_i (the one inside the grey box)? Are they the same?

The computer's local time is used to keep the beacon synchronized to the correct time. The beacon will release (in NIST's case) one pulse for each minute. So there will be a pulse with timestamp

2017-09-27 22:14

and the beacon engine will make sure this pulse is generated and signed and ready before 22:14. At 22:14 (but not a second before), the beacon engine will push this pulse to the database, and immediately afterward, it will be available from the web front end.

The timestamp is really a promise from the beacon operator—it's promising not to release this beacon pulse until the time on the timestamp. Before that, nobody outside the beacon engine should know what's going to be in that pulse.

Q2. What does it mean that the "time service" is remote? Is is cryptographically signed by an external entity ... or is it simply a clock under NIST's control, non-externally auditable?

The beacon engine is a standard Windows box, with a system clock that will drift a little over time. I believe the beacon engine uses NTP to keep its clock synchronized. I don't know anything about any crypto being used for NTP, and really don't understand how NTP works. (I gather there's a mechanism to prevent NTP from moving the clock too quickly.) But the point of the external time service is to keep the beacon engine's clock synchronized with reality.

We could in principle get a very high external precision clock to keep the computer's clock synchronized, if we didn't want to trust NTP.

Q3. How are the "previous pulses" combined together? Are they used as a long concatenation (if yes, of how many), or are they combnied in a kind of chained hash, e.g., PPs_i = SHA512(P_{i-1},PPs_{i-1}), where P_{i-1} would be the pulse at the (i-1)-th iteration?

Every pulse contains the hash of the previous pulse in the previous field. The outputValue field is the hash of the whole rest of the pulse. So the outputValue of pulse

2017-09-27 22:14

is the hash of all the fields in that pulse, which includes the hash of pulse

2017-09-27 22:13

which includes the hash of pulse

2017-09-27 22:12

and so on. These form a hash chain.

The cool thing about a hash chain is that if you alter any record anywhere in the chain, that has to

propagate forward to the last record in the chain.

If you change the localRandomValue in pulse

2017-09-27 22:14

then that must change the outputValue from that pulse. (Otherwise, you've found a hash collision.) And that changes the previous field in pulse

2017-09-27 22:15

which changes ***its*** outputValue, and thus the previous field of pulse

2017-09-27 22:16

and on and on.

This is why we can't alter history—if you've seen pulse

2017-09-27 22:20

and I try to give you an altered version of pulse

2017-09-27 22:14

you'll be able to detect this by running through the hash computations on the sequence of pulses from

2017-09-27 22:14 - 2017-09-27 22:20

Q4. Is the "precommitment value (next record)" (h2_i) only dependent on the "local random value" (h1_i)? Why does it not depend on the previous pulses and the current timestamp?

Just to be clear. If A[T] is the pulse at time T, and A[T+1] is the pulse at time T+1 (one minute later), then we have

A[T].precommitmentValue = hash(A[T+1].localRandomValue)

Now, the reason we only hash A[T+1].localRandomValue, and not the rest of A[T+1], is because A[T+1] will also contain a precommitmentValue. And so will A[T+2], and A[T+3], and so on forever. So the only way for us to get the hash of the whole pulse T+1 into our A[T].precommitmentValue would be to pre-generate every pulse forever before we released the first pulse.

The document I've been writing on combining beacons says that to combine two beacon pulses (A and B) you do:

1. Store A[T-1] and B[T-1] and make sure they arrived before T.

2. Get A[T] and B[T].

3. Verify that

A[T-1].precommitmentValue = hash(A[T].localRandomValue)
B[T-1].precommitmentValue = hash(B[T].localRandomValue)

4. Compute combined output value:

C = hash(A[T-1].outputValue || B[T-1].outputValue || A[T].localRandomValue || B[T].localRandomValue)

Q5, I think I answered above.

*Q6. Lets assume a model where NIST is malicious and can do whatever it wants with the HSM, such as using it as many times as it wants to, without being audited for that. Based on the diagram, it appears that the only randmoness is coming from the RNGs from Intel and Thales, which I guess are NIST local. Lets also assume that the pre-images of the RNG hashes are never recorded or audited, and so NIST can basically choose whatever (fake) hash value it wants. If the previous is correct, then what prevents NIST from computing very quickly (say, in a day), the next 1440*50*365 values (i.e., 1 year's worth of 1-min separated NIST pulses), and then use the upcoming year to brute-force the subsequent pulse? For example, it could make many tries until getting a value with a particular intended structure, e.g., being a tring finalized with 40 zeros? (if the HSM can sign 2^40 values in a year's period).*

That's exactly what will happen. The pulse format limits the attacker's power by forcing him to brute-force inputs to a hash function (the result is the outputValue field). If we are going to have a single service providing beacon pulses, I think it's very hard to overcome this limitation. Worse, the beacon operator will know every pulse forever in the future, because the first thing he'll do when he decides to turn evil is just initalize a PRNG with a good seed, and use that to precompute all localRandom numbers from then on.

That's why we added the precommitmentValue and the support for combining beacons in the new format.

Q7. Would it be possible to have each final pulse be cryptographically timestamped by an external certificatoin authority, and then include said verified timestamp as an input that will affect the next final pulse? Even if this is not possible per pulse, but would be possible once per hour, then the problem mentioned in Q6 would be resolved. Then again, this would mean relying an an external service.

This is what the externalValue field is intended to give us. We could use every 20th Bitcoin block hash as an external input, and even a corrupt beacon operator wouldn't be able to predict or control

anything past each externalValue input.

Unfortuately, that solves one problem, but creates another—we now need a mechanism for sending every 20th Bitcoin block hash into the beacon engine. That means a close-to-realtime thing that's talking directly to the beacon engine, which is a great vector for someone to try to take it over. (There are also some reliability issues there, but the big one is the extra channel from the outside world in.)

I hope this is helpful,

--John

From: "Brandao, Luis (IntlAssoc)" <luis.brandao@nist.gov>
Date: Wednesday, September 27, 2017 at 6:08 PM
To: "Miller, Carl A. (Fed)" <carl.miller@nist.gov>, "Kelsey, John M. (Fed)" <john.kelsey@nist.gov>, "Peralta, Rene (Fed)" <rene.peralta@nist.gov>, "Booth, Harold (Fed)" <harold.booth@nist.gov>
Subject: Re: Process diagram for the randomness beacon

Hi,

I also find the diagrams useful, to better understand the process. I wrote down the relations between several values and that helped me raise several doubts. Below I start by assigning a symbol to each variable, and then proceed with doubts/questions I have after looking at the diagram.

======== Symbols and relations:

- t1_i: timevalue obtained from CPU clock or (?) remote time service, to be used at the i-th iteration
- r1_i: Intel's RNG 512-bit output at the i-th iteration
- r2_i: Thales's HSM's RNG 512-bit output at the i-th iteration
- h1_i = SHA512(r1_i || r2_i): "local random value" at the i-th iteration
- T_i: "timestamp" (as indicated in the grey box) used in the i-th iteration
- Ty_i, Tm_i, Td_i, Th_i: year, month, day, hour at i-th iteration (are all these extracted/parsed from the initial t_i?)
- D_i === Ty_i || Tm_i || Td_i || Th_i (the letter "D" denoting "date" --- useful for abbreviating the concatenation of four values)
- PPs_i: "Previous pulses", as seen at the i-th iteration (is this a concatenation, a chained hash, ...?)
- H_i = SHA512{T_i || PPs_i || D_i || h1_i || h2_{i-1}}: hash of the bulk info at the i-th iteration

- S_i: Signature of H_i, by the HSM at the i-th iteration
- R_i: output "randomness" at the i-th iteration
- P_i = t_i || PPs_i || D_i || h1_i || h2_{i-1} || S_i || R_i : final pulse at the i-th iteration

======== Doubts and questions

Q1. What's the difference between the CPU clock time (t_i) and the "timestamp" T_i (the one inside the grey box)? Are they the same?

Q2. What does it mean that the "time service" is remote? Is is cryptographically signed by an external entity ... or is it simply a clock under NIST's control, non-externally auditable?

Q3. How are the "previous pulses" combined together? Are they used as a long concatenation (if yes, of how many), or are they combnied in a kind of chained hash, e.g., PPs_i = SHA512(P_{i-1},PPs_{i-1}), where P_{i-1} would be the pulse at the (i-1)-th iteration?

Q4. Is the "precommitment value (next record)" (h2_i) only dependent on the "local random value" (h1_i)? Why does it not depend on the previous pulses and the current timestamp?

Q5. I know there is an intention of one pulse per minute. Is NIST trying to synchronize the pulses to the exact minute (i.e., x hours, y minutes, 00 seconds) of convential time (e.g., UTC), or is this irrelevant? Are the values (used in D_i) of "hour", "day", "month" and "year" supposed to be predictable for the purpose of checking, in the future, whether or not a hash was well created? If yes, then how are they defined when we are close to a boundary (change of hour, or day, or month or year)? The unpredictability would come from umpredictable clock skewing of a few miliseconds, e.g., 23h59m59s999ms vs. 00h00m00s001ms when trying to get the exact midnight. This is not a problem if the pulse is for example trying to synchronize with x minutes and 30 seconds. This may overall not be any problem whatsoever if there is not intention of being able to predict those time values for each consecutive pulse.

Q6. Lets assume a model where NIST is malicious and can do whatever it wants with the HSM, such as using it as many times as it wants to, without being audited for that. Based on the diagram, it appears that the only randmoness is coming from the RNGs from Intel and Thales, which I guess are NIST local. Lets also assume that the pre-images of the RNG hashes are never recorded or audited, and so NIST can basically choose whatever (fake) hash value it wants. If the previous is correct, then what prevents NIST from computing very quickly (say, in a day), the next 1440*50*365 values (i.e., 1 year's worth of 1-min separated NIST pulses), and then use the upcoming year to brute-force the subsequent pulse? For example, it could make

many tries until getting a value with a particular intended structure, e.g., being a tring finalized with 40 zeros? (if the HSM can sign 2^40 values in a year's period).

Q7. Would it be possible to have each final pulse be cryptographically timestamped by an external certificatoin authority, and then include said verified timestamp as an input that will affect the next final pulse? Even if this is not possible per pulse, but would be possible once per hour, then the problem mentioned in Q6 would be resolved. Then again, this would mean relying an an external service.

Q8. Is there no sequential integer counter somewhere? Typically that is a facilitating index for external applications, i.e., so that one could easily refer to the i-th pulse for some actual integer "i".

--Luís

From: Miller, Carl A. (Fed)
Sent: Wednesday, September 27, 2017 4:52 PM
To: Kelsey, John M. (Fed); Peralta, Rene (Fed); Booth, Harold (Fed)
Cc: Brandao, Luis (IntlAssoc)
Subject: Re: Process diagram for the randomness beacon

> Hi John --
>
> It looks good to me – it's great to have all of this in one place. My current plan is to try to create some similar diagrams in LaTeX/TikZ, and I might send you all some questions about the individual picture elements. Thanks!
>
> -Carl
>
> ————
> Carl A. Miller
> Mathematician, Computer Security Division
> National Institute of Standards and Technology
> Gaithersburg, MD

> **From:** "Kelsey, John M. (Fed)" <john.kelsey@nist.gov>
> **Date:** Wednesday, September 27, 2017 at 1:36 PM
> **To:** "Miller, Carl A. (Fed)" <carl.miller@nist.gov>, "Peralta, Rene (Fed)" <rene.peralta@nist.gov>, "Booth, Harold (Fed)" <harold.booth@nist.gov>

Cc: "Brandao, Luis (IntlAssoc)" <luis.brandao@nist.gov>
Subject: Re: Process diagram for the randomness beacon

I just used OpenOffice's Draw program. I don't claim any of this is beautiful, but hopefully it gets the idea across anyway.

--John

From: "Miller, Carl A. (Fed)" <carl.miller@nist.gov>
Date: Wednesday, September 27, 2017 at 12:42 PM
To: "Kelsey, John M. (Fed)" <john.kelsey@nist.gov>, "Peralta, Rene (Fed)" <rene.peralta@nist.gov>, "Booth, Harold (Fed)" <harold.booth@nist.gov>
Cc: "Brandao, Luis (IntlAssoc)" <luis.brandao@nist.gov>
Subject: Re: Process diagram for the randomness beacon

Hi John –

Thanks, these will definitely be helpful. What software did you use to draw these, out of curiosity?

-Carl

————
Carl A. Miller
Mathematician, Computer Security Division
National Institute of Standards and Technology
Gaithersburg, MD

From: "Kelsey, John M. (Fed)" <john.kelsey@nist.gov>
Date: Wednesday, September 27, 2017 at 10:49 AM
To: "Miller, Carl A. (Fed)" <carl.miller@nist.gov>, "Peralta, Rene (Fed)" <rene.peralta@nist.gov>, "Booth, Harold (Fed)" <harold.booth@nist.gov>
Cc: "Brandao, Luis (IntlAssoc)" <luis.brandao@nist.gov>
Subject: Re: Process diagram for the randomness beacon

I don't know if these quite work or are useful, but here are some diagrams I've drawn.

--John

From: "Miller, Carl A. (Fed)" <carl.miller@nist.gov>
Date: Tuesday, September 26, 2017 at 12:58 PM
To: "Peralta, Rene (Fed)" <rene.peralta@nist.gov>, "Booth, Harold (Fed)" <harold.booth@nist.gov>, "Kelsey, John M. (Fed)" <john.kelsey@nist.gov>
Cc: "Brandao, Luis (IntlAssoc)" <luis.brandao@nist.gov>
Subject: Process diagram for the randomness beacon

Hi Rene, Harold & John –

Following up on our meeting last week:

I'd like to put together a "process diagram" of the beacon, including all components of the beacon & the information channels between them. (I'd like to do it for "version 2.0".) What's our most up-to-date description of the beacon right now? I have John's notes from last year ("The New NIST Beacon Protocol and Combining Beacons") and a draft of the NISTIR that Rene sent out on June 7th. Can you think of anything else I could use?

(I can also talk with you individually about any aspects that aren't written down yet.)

-Carl

————

Carl A. Miller
Mathematician, Computer Security Division
National Institute of Standards and Technology
Gaithersburg, MD