A Specification And Prototype Of Using PIR For TOR

Alex Seewald

September 16, 2017

1 The PIR Tor Protocol

The descriptor format and microdescriptor formats, described in dir-spec.txt, can optionally include one additional entry related to PIR. By "optionally" it is meant that by default it is assumed that a directory server is not willing to do any PIR. The optional PIR line of an entry, either to a descriptor or microdescriptor, is of the form:

```
pir <will_serve_descriptors> <will_serve_microdescriptors> <port> [<list_of_algos>]
   where
      <will_serve_descriptors> is in {0,1} and expresses whether the given
                               directory server will serve a database of descriptors.
      <will_serve_micro-descriptors> is in {0,1} and expresses whether the given directory
                                     server will serve a database of micro-descriptors.
      <port> is the port on which the directory server will be listening for.
```

The client and server have a configuration, specified in a configuration file rather than at compile-time or as command line arguments, of the pir parameters it is willing to use. This configuration file must be named config.yaml and exist in the tor directory. This file specifies the range of PIR parameters which a tor client running on the given host will accept during negotition, and a separate range of PIR parameters which a tor directory server running on the given host will accept during negotiation. More details about the configuration can be seen in the comments of the configuration file. If one wants to programatically change the configuration, they can use the `pirtor_config.py` program.

After the client connects to the port specified in the previously known descriptor, the pir parameter negotiation occurs. This negotiation consists of just two messages. The client sends the first message, which is a negotiation request message. This negotiation request message is essentially a serialized form of the `negotiate_request_t` data structure in `pir_common.h`. This includes priorities for the different kinds of pir algorithms and cryptographic methods, lower and upper bounds on PIR parameters, and the policy that the server should use for deciding on the exact values they must agree upon. The server waits

for this message and implements the client's policy. If the client and server can agree on parameters, the server sends back a negotiation response message which is essentially a serialized form of the `negotiate_response_t` data structure in `pir_common.h`. If the server cannot find an intersection of parameters, the negotiation response message will have a `pir_opts` value whose agreed field has a false value and whose `disagree_reason` has newline-separated reasons for this failure. If there is disagreement, the server may close the connection with the client.

After negotiation, the client sends a query message per entry and the server waits for these query messages. The number of entries is agreed upon during negotiation, so no separate message is needed to direct the server to start processing the queries. The server then processes the queries, according to the PIR algorithm and with the agreed upon PIR parameters, and sends a result message to the client. The format of these queries and responses depends on the particular PIR algorithm.

After sending the responses, the server may close the connection with the client because there is nothing left to do.

1.1 Remarks

PIR cannot be used to get the descriptors or microdescriptors for the first time. This is because the client must know things about the directory server willing to do PIR, such as whether it is willing to do PIR for the particular thing the client wants to know and if so what port to use. PIR can only be used for updates.

Without PIR, tor saves bandwidth during updates to the consensus by having clients and servers communicate diffs between current knowledge and updated knowledge. The use of PIR to get diffs is outside the scope of this protocol, and possibly not a good idea in principle. The use of PIR for tor will involve identical uses of the protocol during the first use of the client and during updates.

This protocol makes the decision of making the negotiation and the optional part of a descritor or microdescriptor not communicate any information about the servers' ranges of PIR parameters. This has the benefit of sending fewer bytes over the network. This has the drawback that clients may initiate a negotiation and end up discovering that a server is not suitable.

The planning and implementation mostly considers single-server computational PIR, but the protocol described would be suitable for multi-server information theoretic PIR if the client does it with multiple servers.

2 Explanations Of Program Files

2.1 Programs which comprise the pir tor library

- `pir_common.h` : Specifies data structures and constants which are among the names included to use PIR. Can be included by c programs, e.g. tor files, or by c++ programs like the pir tor libraries.
- `pir_common.cpp` : Implements serializers and deserializers of data structures needed to do pir. Serialization is needed because clients and servers need to negotiate pir parameters.
- `pir_client.cpp` : Implements the behavior specific to a pir tor client. This means the client-side of negotiating pir parameters (`pir_client_negotiate`), xpir requests which conform to the pir tor protocol, and logging of relevant performance information.
- `pir_server.cpp` : Implements the behavior specific to a pir tor server. This means the server-side of negotiating pir parameters (`pir_server_negotiate`), xpir responses which conform to the pir tor protocol, and logging of relevant performance information.
- `parse_desc.py` : Parses a file which is a concatenation of tor descriptors. It separates them out and saves them in a database suitable for XPIR.
- `parse_microdesc.py` : Parses a file which is a concatenation of tor microdescriptors. It separates them out and saves them in a database suitable for XPIR.
- `parse_common.py` : functionality shared by `parse_desc.py` and `parse_microdesc.py`

2.2 Programs which use the pir tor library to run experiments

- `simulation_experiment.ipynb` : an ipython notebook which illustrates the feasibility of pir tor by running it. It runs the simulation server and simulation client programs with a parameter sweep, and then it visualizes the timing information.
- `simulation_common.cpp` : code shared by the client-side and server-side of experiments. this is not part of the pir tor library.
- `simulation_server.cpp` : a program which will wait for a client and repeatedly listen for commands. If the command is related to a full upload, it will send the full database. If the command is related to pir, it will start executing the pir tor protocol. If the command says to quit, the server will stop running.

- `simulation_client.cpp` : a program which connects to a server and sends commands to the server. This number of commands is either specified as a command line argument or otherwise it will send just one command. These commands are read in over stdin.

2.3 Remarks

All communication between client and server does not use predecided send and recieve functions. Instead the interface has function pointers, and programs using pir tor must pass in their own communication functions. See `simulation_common.cpp` for an example. Even though pir tor is implemented in c++ (mostly to deal with PIR libraries conveniently) it can export functions to c programs by being compiled in a particular way, as done in the Makefile. This is important because tor is a c project.

3 Compilation,Installation,Dependencies

The depencies are XPIR for the heavy lifting of the PIR, yaml-cpp, for dealing with the configuration, a python installation containing the modules used in the parsing programs, and a compiler willing to use the C++11 standard.

The Makefile is used for all compilation. `make pirtor` builds pir tor as a library, suitable for use in tor or elsewhere. `make client` makes a client using pir tor for experiments. `make server` makes a server using pir tor for experiments.

4 On the Roadmap but not yet implemented

- PIR libraries other than XPIR.
- Use of the XPIR optimizer. For now `INTERSECT_OPTIMAL` won't lead to what is intended, and instead the server will choose arbitrary parameter values within the ranges.
- Modifying TOR to actually use pir tor.